Variability and Customization of Simulator Products

A Product Line Approach in Model Based Systems Engineering

Linköping Studies in Science and Technology.
Dissertation No. 1427

Variability and Customization of Simulator Products

A Product Line Approach in Model Based Systems Engineering

Henric Andersson

Linköping University

Department of Management and Engineering
Division of Machine Design
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2012

Variability and Customization of Simulator Products
A Product Line Approach in Model Based Systems Engineering

Linköping Studies in Science and Technology. Dissertation No. 1427
ISBN 978-91-7519-963-4
ISSN 0345-7524

henric.andersson@liu.se
www.iei.liu.se/machine
Division of Machine Design
Department of Management and Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Printed in Sweden by LiU-Tryck Linköping, 2012

Abstract

AIRCRAFT DEVELOPERS, like other organizations within development and manufacturing, are experiencing increasing complexity in their products and growing competition in the global market. Products are built from increasingly advanced technologies and their mechanical, electronic, and software parts grow in number and become more interconnected. Different approaches are used to manage information and knowledge of products in various stages of their lifecycle.

"Reuse" and "Model Based Development" are two prominent trends for improving industrial development efficiency. The product line approach is used to reduce the time to create product variants by reusing components. The model based approach provides means to capture knowledge about a system in the early lifecycle stages for usage throughout its entire lifetime. It also enables structured data management as a basis for analysis, automation, and team collaboration for efficient management of large systems and families of products.

This work is focused on the combination of methods and techniques within;

- modeling and simulation-based development, and
- (re)use of simulation models through the product line concept.

With increasing computational performance and more efficient techniques/tools for building simulation models, the number of models increases, and their usage ranges from concept evaluation to end-user training. The activities related to model verification and validation contribute to a large part of the overall cost for development and maintenance of simulation models. The studied methodology aims to reduce the number of similar models created by different teams during design, testing, and end-user support of industrial products.

Results of the work include evaluation of a configurator to customize and integrate simulation models for different types of aircraft simulators that are part of a simulator product family. Furthermore, contribution comprises results where constraints in the primary product family (aircraft) govern the configuration space of the secondary product family (simulators). Evaluation of the proposed methodology was carried out in cooperation with the simulator department for the 39 Gripen fighter aircraft at Saab Aeronautics.

När komplexiteten ökar, kommer Systems Engineering och spökar.

- Mehdi Tarkian

Sammanfattning

FLYGPLANSTILLVERKARE LIKSOM andra industrier inom utveckling och tillverkning, hanterar ökande komplexitet i sina produkter och upplever en större konkurrens på den globala marknaden. Produkter byggs från allt mer avancerad teknologi. Ingående delar av mekanik, elektronik och mjukvara växer i antal och blir allt mer integrerade. Olika metoder används för att hantera information och kunskap om produkter i olika steg av dess livscykel.

”Återanvändning” och ”Modellbaserad utveckling” är två tydliga trender för att öka effektiviteten inom industriell utveckling. Produktfamiljer används för att minska ledtider när man skapar varianter av produkter genom att återanvända färdiga komponenter. Modellbaserade metoder ger möjlighet att tidigt i livscykeln samla kunskap om ett system för att användas under hela systemets livstid. De ger också strukturerad hantering av data som grund för analys, automatisering och samarbete mellan utvecklingsteam, vilket är en förutsättning för effektiv hantering av komplexa system och produkter.

Detta arbete är fokuserat på en kombination av metoder och tekniker för;

- utveckling som baseras på modellering och simulering, och
- (åter)användning av simuleringsmodeller.

Med ökande beräkningsprestanda och effektivare metoder/verktyg för att bygga simuleringsmodeller så ökar antalet modeller och deras användning spänner allt från konceptvärderingen till utbildning av slutanvändare. Arbetet med verifiering och validering av simuleringsmodeller utgör en stor del av deras totala utvecklings- och underhållskostnader. De studerade metoderna syftar till att minska antalet liknande modeller som hanteras av olika team för olika syften, som till exempel; utveckling, verifiering och som stöd för slutanvändare.

Resultat av arbete inkluderar utvärdering av en konfigurator för att välja, integrera och anpassa simuleringsmodeller för olika typer av flygplanssimulatorer i en simulatorproduktfamilj. Dessutom bidrar arbetet med en metodik där begränsningarna i den primära produktfamiljen (flygplan) begränsar konfigurationsutrymmet för den sekundära produktfamiljen (simulatorer). Utvärdering av den föreslagna metoden har genomförts i samarbete med simulatoravdelning för flygplan 39 Gripen på Saab Aeronautics.

Everything must be made as simple as possible. But not simpler.

- Albert Einstein

Model Based Development!? Wouldn't it be better with Reality Based Development?

- August Andersson

Acknowledgments

THE WORK PRESENTED in this dissertation was carried out as an industrial PhD project at the Division of Machine Design at the Department of Management and Engineering (IEI) at Linköping University. Saab Aeronautics was the industrial partner and provided the opportunity to relate the work to an innovative industrial environment.

There are several people I would like to thank for their support and advice. First, I want to thank Erik Herzog for his guidance, academic views, and for being an inspiring co-author of some of the publications. Thanks also to the industrial sponsors at Saab, especially Anders Pettersson and Stefan Andersson who involved me in the *MBSE program* at Saab, and for giving me the opportunity to start the research journey.

I am also grateful to my supervisors at the university, Prof. Petter Krus and Prof. Johan Ölvander for your guidance and support. To Olof Johansson and Björn Lundén I show appreciation for supervision during the first part of the work.

My managers at Saab, Ulrik Pettersson and Henrik Pettersson; you gave me important support and found time when I needed it. A special thank goes to Sören Steinkellner who inspired me and often took me down from the 'abstract heights'. At Saab, there are many colleagues and co-authors that I want to thank; Ingela Lind, Magnus Carlsson, and many more. Thanks also to Lars Karlsson, Magnus Jomander, Kristoffer Johansson, Robert Lindohf and all of you in the *EMMA team* and at the *Saab Simulator-Center* who provided me with challenges, knowledge, and data for this research. I also wish to thank members of the *Machine Design* and *Fluid and Mechatronic Systems* divisions for your friendship and valuable feedback. I am also grateful to the ProViking research school and all inspiring PhD students.

The results reported in this dissertation were inspired by and based on research funded by: the Swedish Governmental Agency VINNOVA's National Aviation Engineering Research Programs, NFFP 2006-02705 and NFFP 2009-01359; the European Community's Sixth Framework Programme (FP6/2006-2009) SPEEDS (contract n° 033471); and the European Community's Seventh Framework Programme (FP7/2007-2013) CRESCENDO (grant agreement n° 234344).

Finally, I thank my family; Christina, August, Axel, and Bertil for being so patient during the periods of struggling and writing.

Henric Andersson

Borensberg, March 2012

List of appended papers

THIS DISSERTATION is based on the following seven papers, which are appended and will be referred to by their Roman numerals. The papers are included in their originally published state except for changes in formatting and correction of minor errata.

[I] Andersson, H., Herzog, E., Johansson, G. & Johansson, O. (2010). Experience from introducing Unified Modeling Language/Systems Modeling Language at Saab Aerosystems. *Systems Engineering, 13*: 369-380. doi: 10.1002/sys.20156

[II] Lind, I. & Andersson, H. (2011). *Model Based Systems Engineering for Aircraft Systems – How does Modelica Based Tools Fit?*. In Proceedings of the 8th International Modelica Conference. Dresden, Germany. doi: 10.3384/ecp11063856

[III] Andersson, H. (2010) *Variability and Configuration Principles for Simulation Models in Product Line Development*. In Proceedings of the 7th European Systems Engineering Conference, EuSEC 2010, Stockholm, Sweden.

[IV] Andersson, H., Steinkellner, S. & Erlandsson, H. (2010). *Configuration Management of Models for Aircraft Simulation*. In Proceedings of the 27th International Congress of the Aeronautical Sciences, ICAS. Nice, France.

[V] Carlsson, M., Andersson, H., Gavel, H. & Ölvander, J. (2012). *Methodology for Development and Validation of Multipurpose Simulation Models*. In Proceedings of the 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Nashville, TN, USA.

[VI] Andersson, H., Carlsson, M. & Ölvander, J. (2011). *Towards Configuration Support for Collaborative Simulator Development: A Product Line Approach in Model Based Systems Engineering*. In Proceedings of the 20th IEEE International Conference on Collaboration Technologies and Infrastructures. Paris, France. doi: 10.1109/WETICE.2011.74

[VII] Andersson, H., Herzog, E. & Ölvander, J. (2012). Experience from Model and Software Reuse in Aircraft Simulator Product Line Engineering. Conditionally accepted for publication in *Information and Software Technology*.

Out of the seven papers, Andersson is the main contributing author of papers [I], [III], [IV], [VI], and [VII].

The following papers are not appended but constitute a part of the background:

[VIII] Andersson, H. & Sundkvist, B.G. (2006). *Method and Integrated Tools for Efficient Design of Aircraft Control Systems*. In Proceedings of the 25th International Congress of the Aeronautical Sciences, ICAS. Hamburg, Germany.

[IX] Steinkellner, S., Andersson, H., Krus, P. & Lind, I. (2008). *Hosted Simulation for Heterogeneous Aircraft System Development*. In Proceedings of the 26th International Congress of the Aeronautical Sciences, ICAS. Anchorage, AK, USA.

[X] Hallberg, P., Andersson, H., Nåbo, M. & Krus, P. (2008). *Modular sustainable light multi-purpose vehicle*. In Proceedings of the 3rd European Ele-Drive Transportation Conference - EET-2008. Geneva, Switzerland.

[XI] Johansson, O., Andersson, H. & Krus, P. (2008). *Conceptual Design Using Generic Object Inheritance*. In Proceedings of the ASME International Design Engineering Technical Conference and Computers and Information in Engineering Conference 2008, IDETC/CIE. Brooklyn, N. Y., USA.

[XII] Andersson, H., Weitman, A. & Ölvander, J. (2008). *Simulink as a Core Tool in Development of Next Generation Gripen*. In Proceedings of Nordic Matlab User Conference 2008. Stockholm, Sweden.

[XIII] Steinkellner, S., Andersson, H., Gavel, H. & Krus, P. (2009). *Modeling and simulation of Saab Gripen's vehicle systems*. AIAA Modeling and Simulation Technologies Conference. Chicago, IL, USA.

[XIV] Herzog, E. & Andersson, H. (2009). *Initial Experience in Contracts Based Systems Engineering*. In Proceedings of the 19th annual international symposium of International Council on Systems Engineering, INCOSE. Singapore.

[XV] Herzog, E., Andersson, H. & Hallonquist, J. (2010). *Experience from Introducing SysML into a Large Project Organisation*. In Proceedings of the 20th annual international symposium of International Council on Systems Engineering, INCOSE. Chicago, IL, USA.

[XVI] Steinkellner, S., Andersson, H., Gavel, H., Lind, I. & Krus, P. (2010). *Modeling and Simulation of Saab Gripens Vehicle Systems: Challenges in Processes and Data Uncertainties.* In Proceedings of the 27th International Congress of the Aeronautical Sciences, ICAS. Nice, France.

Abbreviations

a/c	Aircraft
CAD	Computer-Aided Design
CI	Configuration Item
CM	Configuration Management
ECS	Environmental Control System
ECU	Electronic Control Unit
FMI	Functional Mockup Interface
H/W	Hardware
HILS	Hardware In-the-Loop Simulation
KBE	Knowledge Based Engineering
M&S	Modeling and Simulation
MBD	Model Based Development
MBSE	Model Based Systems Engineering
PDM	Product Data Management
PVM	Product Variant Master
S/W	Software
SCM	Software Configuration Management
SPL	Software Product Line
SysML	Systems Modeling Language
XML	EXtensible Markup Language
XSLT	EXtensible Stylesheet Language

Table of contents

Appended papers

1 Introduction

MOST PRODUCTS and systems evolve towards higher degrees of complexity consisting of more advanced technologies with integrated mechanical, electronic, and software parts. Products are increasingly defined in families, allowing variants and versions of similar products to be offered to customers. Model Based Development (MBD) is one approach to manage large, complex systems and families of products. It enables structured data management to serve as a basis for analysis, automation, and collaboration for example. MBD also provides a way, through the virtual product and system simulations, to gain insights and understanding for engineers and others involved in the development, verification, and maintenance of the products. Validated simulation models from systems development can be reused in simulator products to achieve training of end users with high fidelity simulations in a virtual environment.

1.1 Background

Organizations developing and manufacturing high-end products have for decades relied on engineering methods that are based on some kinds of models; and the use of model based methods are increasing. In the management of large, complex products and their related information, different product parts may in different degrees be based on the modeling approach. The development lifecycle for a specific system (or some additional system functionality) starts with a verbal formulation and evolves through system design to rollout for customer delivery and usage.

There are obviously different ways to execute the development, depending on the choice of engineering means, as illustrated in Figure 1. Maturity of a system and its functionality is reached through several activities, where modeling, analysis, and simulation “get the function to mature” more rapidly compared to traditional document-centric methods. Parts of both verification and validation are performed earlier with the support of models and simulations, providing greater possibility to improve poorly stated requirements early and to find defects/non-optimal designs.

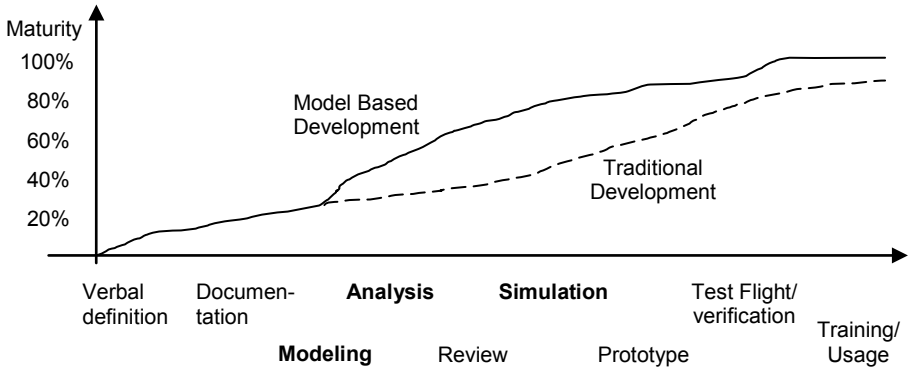

Figure 1. The basic idea of model based development is that a system's maturity increases faster by enabling analysis and simulations of the system from an early stage. The models support reuse of knowledge during initial operation of the system and throughout its entire lifetime.

To support development of complex products, several methods and languages have been proposed in the literature and evaluated in industry, as reported in Mar [1992] and Stevens [1998]. In recent years, there has been much focus on MBD as means for managing complexity by improving specification clarity, consistency, and validation support, see Alford et al. [1992], Oliver, Kelliher and Keegan [1997], Wymore [2002], France and Rumpe [2007], Weilkiens [2008], and Friedenthal, Moore and Steiner [2011].

Another prominent trend is *reuse* of artifacts such as design solutions and system/software components, for reducing development and production cost while still offering a range of similar products to the market. From the 1980s and onward, software reuse is one of the main reasons for increased productivity for software-intensive systems [Boehm 2006]. A set of similar products that share components and are developed and manufactured within a defined architecture constitutes a *product family* or a *product line*. Companies who provide the market with products in several variants and versions increasingly rely on the *product line approach* with explicitly defined *product line architecture*.

Descriptions of approaches and methods for efficient utilization of reuse through product lines and customization support have been provided in literature. For products in general, see Simpson [2006], Ullrich and Eppinger [2008], and Hvam, Mortensen and Riis [2008]. A somewhat different kind of product line approach has been defined for software intensive systems/products, called Software Product Lines (SPL); see Weiss and Lai [1999], Clements and Northrop [2002], van der Linden, Schmid and Rommes [2007], and Jarzabek [2007].

Figure 2 illustrates some basic parts of a product line setting:

- reusable core assets/components with variations
- decisions about variation for specific products
- variation binding through an instantiation process

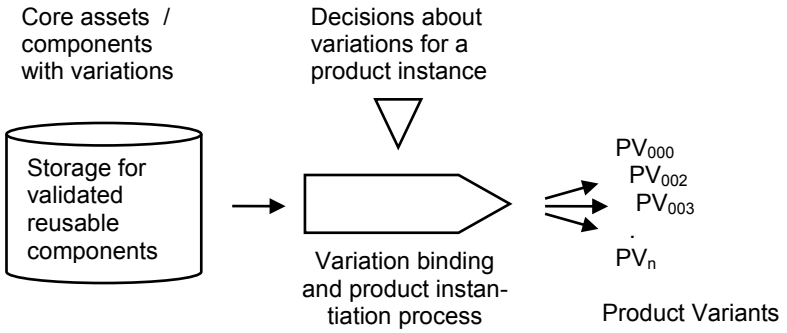

Figure 2. The basics of a product line architecture where existing components are used in a product instantiation process to create customized products. The product instantiations are explicitly governed by configuration decisions. Based on Krueger [2004].

When the *model based* and *product line* approaches are utilized in combination, there is a need for Modeling and Simulation (M&S) of a large number of product variants. While computational performance increases, modeling techniques and tools mature and engineers become more skilled in M&S, the number of models increases and management of the models themselves defines a new domain of knowledge. Praehofer [1996] discusses different approaches known from software engineering, in particular the object-oriented technique, for enhancement of reusability in large-scale simulation systems. Principles for reuse of simulation models and code are covered by Praehofer [1996], Harrison, Gilbert, Jeffrey, Lauzon, and Lestage [2004], Matharu [2006], and Nagy and Cleophas [2011]. None of these takes, however, clearly up the relationship between the models and the products they are supposed to represent. A knowledge gap is identified for the variability and reusability of models with respect to variants and versions of products and models.

The application example used in this work is simulation models and simulators for the Saab 39 Gripen aircraft product family. There are specific characteristics of a simulation model (family) for complex products such as aircraft:

- the properties of system safety require a robust methodology, for example change control, traceability, verification and validation of models;
- a simulation model for an aircraft consists of several unique sub-models developed by different teams, during different times, using different modeling techniques; and
- most of the simulation models are *representations* of another product family; viz the aircraft.

The simulation components (models) do not in general have the same functionality as the represented components (a/c equipment). The models are enhanced with for example fault-simulation functions, but may be simplified in other respects. This implies that variations and combinations of the simulation models are partly constrained or guided by the variability rules of the aircraft's components and functions. To increase the potential for reuse, the simulation models are designed to be included in different kinds of simulators, i.e., they are *multipurpose* models.

Saab Aeronautics has a long history of simulator development, mostly for in-house usage, which is for early validation, development, and system verification. During the last decade, development and delivery of training simulators for the Saab 39 Gripen has become a business. It is now possible to support customers with up-to date simulator versions synchronously with deliveries of new system versions of the aircrafts. The simulators are e.g. based on embedded software used in aircraft. This method is denoted the 'design once approach' and its aim is to increase the conformance and quality of simulator updates. The intention is to continue and to expand the 'design once approach', by developing methods to make coordinated updates in different simulators for several purposes.

1.2 Problem definition

Simulation models may be used in different contexts with specific objectives. Development of the models has often been performed by different teams, which has resulted in a wide range of types and variants. These models are used in different simulation environments, based on different computer languages, and specified for specific operating systems [European Space Agency 2003]. Explicit usage contexts of simulation models in the aircraft industry, and especially within Saab Aeronautics are [III]:

- *Development*; as a tool, for example for analysis and optimization of the design
- *Verification*; is the system safe? Does it satisfy the specified requirements?
- *Training*; to improve the operating skills of end users.

Assumptions and basic elements of the problem definition are as follows: There exists a defined product family (e.g. a set of aircraft variants). This product family is modeled in respect of:

- *Configuration;* for certification, delivery, and maintenance in a product data management context (results in a product configuration model)
- *Behavior;* for development, verification, and training in a simulation context (results in a set of simulation models).

Assume further that the unique sub-models are *configurable*, meaning that one model may represent different variants of the product parts/components/subsystems or its environment. The configurable models are developed, validated, reused, and maintained within the same lifetime as the products.

Each simulation model included in any simulation system can be viewed as an interchangeable component. These components are stored in a library for easy access and inclusion in different simulator configurations and must be able to be configured in at least three dimensions: *representation, usage,* and *implementation* in the following manner:

- A component will *represent* some part of the simulated product or its environment
- A component will be *used* in a context (development, verification or training)
- A component will be *implemented* for a specific simulation platform.

There are no proven concepts, verified methods, or mature supporting tools found to handle configurable simulation models in the described context.

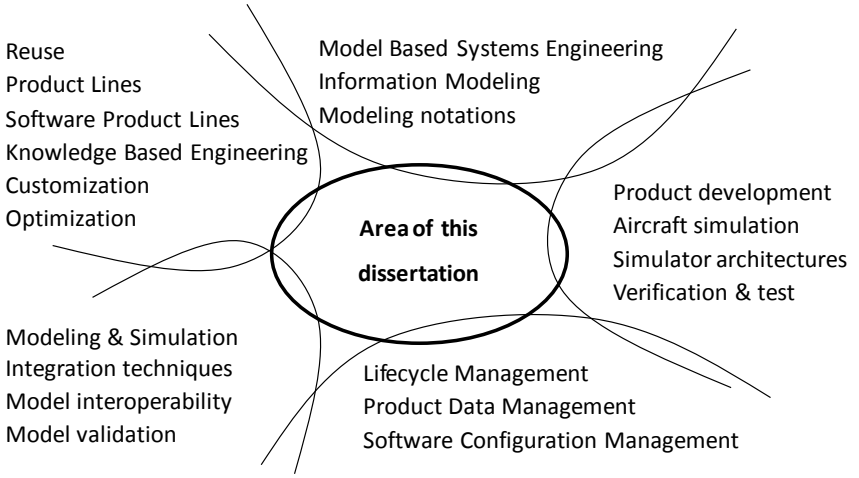

Figure 3. Problem area and related research fields / engineering practices.

The subject of this dissertation covers some traditional areas of research and it is therefore of an interdisciplinary nature. The focus of the research relates to methods, techniques, and engineering practices as shown in Figure 3.

1.3 Industrial objectives

The purpose of the work from an industrial perspective is to develop methods for reuse of simulation models through knowledge about variability and compatibility in order to customize simulator instances. The objectives are related to quality and response-time in the process from a customer request (external or from company-internal customers) to delivery of an a/c simulator. Special focus areas in this work from an industrial perspective relate to methodologies to support:

- Transformation of a product portfolio into a product line by extracting knowledge from legacy simulation software
- Design and implementation of sustainable simulation models for long-term reuse potential
- Handling of non-functional properties and quality attributes
- Product line engineering for large, complex simulator systems
- Alignment with industry-standard Product Data Management (PDM) and Software Configuration Management (SCM) systems
- Alignment with emerging standards and ongoing industrial related research in the fields.

A central part of the dissertation is thus to formalize the engineering challenges that arise when a model based approach is used together with the reuse of models within a large organization that is responsible for complex industrial products.

1.4 Dissertation outline

This dissertation is written in an integrated-paper format and consists of an introductory summary and seven appended papers. The remaining part of the introductory summary is outlined as follows: Chapter 2 covers the research methods. Chapter 3 introduces the theoretical frameworks of Model Based Development and Model Based Systems Engineering. In chapter 4 Knowledge Based Engineering, configuration principles, and available methods for simulation of products on an industrial scale are described. An introduction to the industrial application example is provided in chapter 5. Finally, chapters 6 and 7 cover results and discussion/conclusions respectively, and chapter 7 ends with identified areas for future work.

2 Research methods

THE RESEARCH REPORTED herein is classified as applied research as it was performed with relatively mature systems engineering techniques and in close relation to an industrial organization. This chapter covers the research questions, research environment, related projects, research approach, and contributions of the research work.

2.1 Research questions

By reuse of simulation models and integrating a configurator system in the simulator development environment, providing input to the simulator build and installation processes, it should be possible to deliver customized simulation kernels with specified configurations in a shorter time.

Research question

"Are the principles of Product Customization applicable for modular simulation systems in a software intensive product line context?"

Further sub-questions:

- To what degree is it possible to modularize the simulation model including parameter sets and other simulation artifacts?
- What kinds of variation techniques for simulation model variability, including embedded software, are applicable in the product instantiation process?
- How should compatibility constraints for simulation models be specified to be maintainable in a large-scale product line?
- How to specify and build simulation configurator systems based on Product Data Management (PDM) where environment models are not part of the product definition?

Areas to be dealt with during the research are oriented towards simulation models of vehicle systems, safety critical aspects, and existing products with legacy components. For an overview of the vehicle systems area, see Moir and Seabridge [2004]. Areas beyond the scope of the present work, and therefore not focused upon, include code-generation techniques, multi-core computation, formal methods, early product concepts, and consumer oriented mass customization.

2.2 Research environment and related projects

The work was performed part-time over seven years and it evolved in focus over time from a broad perspective on Model Based Development (MBD) to a narrow focus on software product lines and further to variability and customization of simulator products. There were five research projects related to this work, all within the area of MBD, but with slightly different focus. The research program that financed each project together with other key data is included in Table 1.

2.2.1 System Engineering and Computational Design

Studies of how a product's requirements over its lifecycle can be translated into requirements concerning components were performed during the course of the project. These included development of models and calculation modules used in simulation and optimization at product level. Tools were built in the form of demonstrators in order to show how the methods work in practice. Being part of the project was a good way to get introduced to the problem domain and to an academic way of thinking. Linköping University was the project leader. An introductory paper [Andersson & Sundkvist 2006] on industrial experiences from model based flight control modeling was also published.

2.2.2 Speculative and Exploratory Design in Systems Engineering

In the research project SPEEDS (Speculative and Exploratory Design in Systems Engineering) [Engel, Winokur, Döhmen & Enzmann 2008] a great deal of the work was performed in an industrial context together with aerospace industries, for example Airbus and Saab. Coordinator of the project was Airbus (Germany). Needs for large-scale modeling and analysis were collected and analyzed, and an engineering environment was developed and validated from requirements. The focus was on avionics development in early phases. A meta-model for Heterogeneous Rich Components (HRC) was developed as a means for contract based modeling, tool integration, and analysis capabilities. Representing the needs of an industrial project partner (Saab), validation of the environment with tools for contract-based modeling was included in the task. Experience from contract-based modeling was reported in Herzog & Andersson [2009].

2.2.3 Modeling technique for avionics design

In this project, a prototype of the FM-design tool (function-means-tree) was developed, see [Johansson, Andersson & Krus 2008]. Different approaches to model development and integrated (hosted) simulation were evaluated, see papers [Andersson, Weitman & Ölvander 2008] and [Steinkellner, Andersson, Krus & Lind 2008]. A summary of the first phase of the work was presented and published through a Licentiate Thesis [Andersson 2009]. Saab Aerosystems (now Saab Aeronautics) was head of the project.

Table 1. Research projects participation. Work done as part of the dissertation within each project.

Program / Project	Project name	Years	Methods & techniques	Research as part of the dissertation
Swedish Foundation for Strategic Research, ProViking / **SECD**	System Engineering and Computational Design	2003-2007	▪ system simulation ▪ engineering design optimization	Assessment of M&S techniques (DES, ODE, DAE) and optimization.
European Union Sixth Framework Programme / **SPEEDS**	Speculative and Exploratory Design in Systems Engineering	2006-2010	▪ hosted simulation ▪ analysis techniques, HRC ▪ tool integration ▪ SysML	Specification of needs for large scale modeling environments. Validation of contract based modeling and tool integration.
Swedish National Aeronautics Research Programme 4 / **NFFP4**	Modeling Technique for Avionics Design	2007-2008	▪ product modeling ▪ function-means tree ▪ hosted simulation ▪ Modelica, SysML	Development of the FM design tool. Comparison of M&S techniques. Definition of the Modeling Domain framework.
European Union Seventh Framework Programme / **CRESCENDO**	Collaborative & Robust Engineering using Simulation Capability Enabling Next Design Optimisation	2009-2012	▪ virtual testing ▪ tool integration ▪ collaborative engineering ▪ product data handling ▪ Modelica, FMI	Requirements elicitation for collaborative engineering in aerospace development including certification. Model integration and scale-up means. Architectures for virtual testing.
Swedish National Aeronautics Research Programme 5 / **NFFP5**	Heterogeneous Modeling and Simulation technique	2009-2012	▪ large scale simulation ▪ knowledge engineering ▪ product line engineering ▪ XML and schemas; XSD ▪ feature modeling ▪ constraint programming	Integration of simulation models from different domains. Compatibility and configuration support for large scale simulation. Validation of configurator prototype.

2.2.4 CRESCENDO

CRESCENDO [2010] is a large project divided into several subprojects and use-case definitions. It has a focus on collaborative engineering with support from standardized methods/tools and covers a large part of the product lifecycle including virtual testing and support for aircraft certification. Evaluation of simulation techniques with Modelica based tools was reported in paper [II]. A paper on methodology for development and validation of multipurpose simulation models has been published [V]. Airbus (France) is coordinator of the project.

2.2.5 Heterogeneous Modeling and Simulation technique

A good opportunity was given to finalize the PhD studies/dissertation through the NFFP5 project Heterogeneous Modeling and Simulation technique. Recognition of the model management problem was made in earlier projects and here the focus was on solutions to the "simulation model variation and customization problem". Problem definitions were published in papers [III] and [IV]. In this project, a research approach with shorter increments was used in order to conduct Plan-Act-Observe-Reflect [Williamson 2002] cycles with practitioner interaction. A "towards" paper [VI] was published, reporting on initial findings from the configurator prototype development. The final report from the industrial setting and the validation results are provided in paper [VII]. Saab Aeronautics was head of the project.

2.2.6 From a broad to a narrow research field

The research studies were performed in two major phases with a different broad focus. The first phase (up to the licentiate thesis), had a broad focus and in the second phase the research field was substantially narrower.

- Broad research focused on literature studies, assessment of industrial state of the art (SECD), but also validation of the hosted simulation technique (SPEEDS and NFFP4). The results were summarized in a Licentiate thesis in March 2009, [Andersson 2009]
- Narrower research focused on large-scale simulation, the management of simulation models, and validation of a configurator prototype for customization and integration of simulation models. The second phase was connected to CRESCENDO and NFFP5.

2.3 Research approach

The research reported in this dissertation was conducted as an industrial PhD project, which means that the researcher has a strong relation to the industry and the research is inherently classified as applied research.

2.3.1 Relationship between development and research

Similar methods are used for development of products and for research on development methods. This research work does not penetrate techniques used in products (such as choice of material, components, or technical solutions) in depth, but covers techniques and tools that can support engineers in building and using models for product development. This can be viewed as the meta^2 activity according to Muller [2011], see Figure 4.

Meta^0 is the actual systems development and product creation, meta^1 is the development methods^{1}used to create and manage the product. Meta^2 is the focus of this dissertation; a study of available methods and tools for the meta^1 activities. At meta^3, the research method for comparing tools and methods at meta^2 is defined.

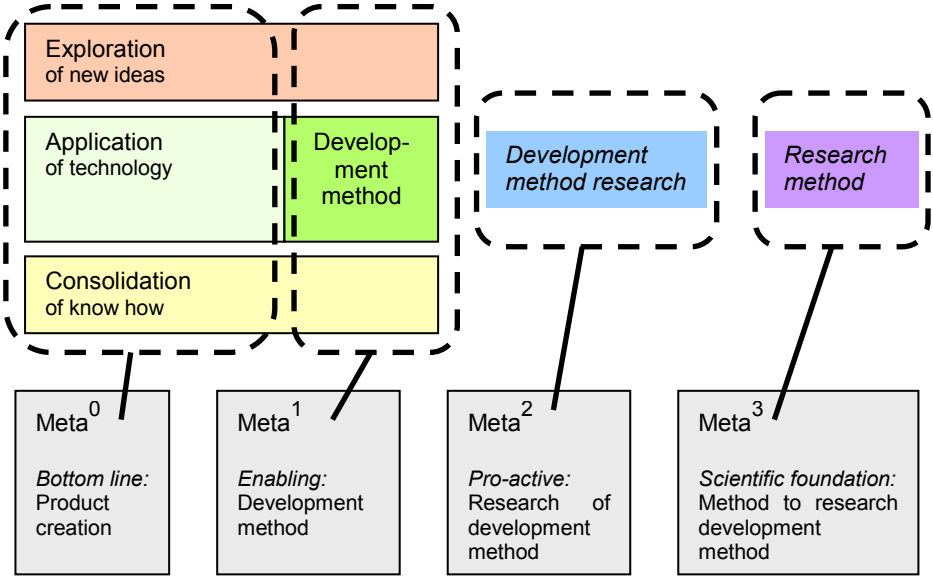

Figure 4. Research method formulation for method research adopted from Muller [2011]

Methods traditionally used for research can also be applied to support development, see [Borg 2009], where the traditional Case Studies research method was used as a method in systems development. In this work, the systems/software development method known as Scrum, [Schwaber 1995; Kniberg 2007] is used to support validation of the research results. This use of Scrum is further described below.

2.3.2 Description of the research

The first part of the research was carried out as a survey of existing methods and tools within model based development. *Modeling Domains* are defined for classification of modeling techniques, and how the tools within the domains can be interconnected, e.g. using the *hosted simulation* technique. An implementation and evaluation of hosted simulation was carried out at the department for vehicle systems at Saab.

In the second part, which focuses on model reuse, the problem area was defined through inventories and interviews at Saab, and by means of literature studies. It was decided to implement means for reuse of simulation model in a configurator prototype, and evaluate the results in interaction with the simulator group at Saab. Studies of *build processes* (checkout, compile, link, and instantiate) for software intensive systems show the importance of *binding time* [Krueger 2004], which is also true of integrated simulation models. Features can be bound (selected/deselected) during different parts of the

build process. Binding describes where in the process a decision controls the inclusion of a specific feature.

Analysis clarifies in what part different configuration information is needed. Information models of product representation in two domains, the Product Data Management (PDM) domain, and the simulation domain are elaborated. Information from interviews and quantitative data is collected and analyzed in order to create an information model. The configurator prototype is developed based on the information models and collected data. Support for the final design of configuration rules and customization solution is based on technology from Sales Configuration Systems, using constraint-based feature modeling and descriptive constrain programming.

The application example in the work consists of simulation models in simulators for the Saab 39 Gripen lightweight fighter aircraft. The Saab Gripen project is analyzed prospectively and typical activities performed at the industry site are:

- inventory of aircraft variants
- inventory of models and their properties from a customization perspective
- inventory of simulator variants
- analysis of the relationships between aircrafts, models, and simulators.

Inventories are made using both quantitative and qualitative methods. The Scrum retrospective method is used to validate the results.

2.3.3 Industry-as-laboratory

This is an industrial PhD dissertation with a focus on industrial large-scale application of both emerging and proven techniques, and with prototype development included. The approach may be defined as "Industry-as-laboratory", as described by Potts [1993]; see Figure 5. By conducting the research in close interaction with industry and iteratively implementing and evaluating results in the industrial setting, new knowledge from the industrial context is fed back to the researcher. Evaluations of suggested and tested solutions are typically made by prototyping.

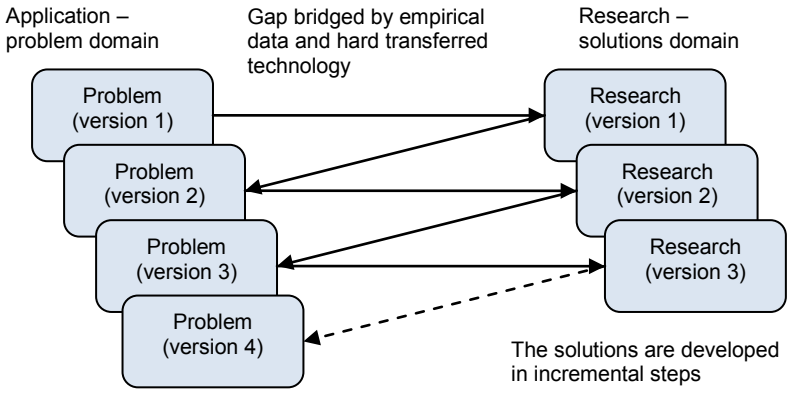

Figure 5. Industry-as-laboratory research approach [Potts 1993]

Industry-as-laboratory stresses both the industrial and the academic relevance of the research, meaning that it should provide useful results to academia (by gaining new knowledge) and to industry (improved quality and reduced cost & time). This approach is qualitative and can be classified as a kind of Interactive Research because of the strong interaction between researcher and practitioner. It typically involves a Plan-Act-Observe-Reflect cycle [Williamsson 2002], which is similar to the Scrum methodology. This is the reason for choosing an approach to conduct systems development research (with influence from the interactive research method) in the industrial team where Scrum had been used for approximately three years when the research started.

2.3.4 Scrum as method to support interactive research

Scrum can be described as an agile, iterative, and incremental development method with structured planning and feedback by means of retrospectives. It has a set of practices and components and the predefined roles of the method are:

- *ScrumMaster*, maintains the workflow and lead the daily meetings
- *Product Owner*, represents the stakeholders and prioritizes incoming work
- *Team*, performs the actual analysis, design, implementation, testing, documentation, and live demonstration of progress achieved.

During each *sprint*, typically a period of a few weeks, the team creates a product increment. The set of features that go into a sprint come from the *product backlog*, which is a prioritized set of tasks to be done. How many backlog items go into the sprint is determined during the sprint-planning meeting. During this meeting, the Product Owner participates and finalizes the priority for the coming sprint. The team determines how much they can commit to complete during the sprint, and records this in the *sprint backlog*. During a sprint, the sprint backlog is fixed, meaning that the requirements are frozen for that sprint. Development is *time-boxed* such that the sprint ends on time. After each sprint, the team demonstrates completed work and a *sprint retrospective* is performed where the team goes through good and bad experiences from earlier work and agrees on improvements to implement.

In the research setting, sprint retrospectives were used as a basic method to collect qualitative data from the team. A questionnaire with issues relevant for validation was prepared prior to each increment, and answers were collected and stored for analysis and evaluation. The outcome of the questionnaire also influenced the planning of the coming iterations. This way of creating and using feedback is described in Salo and Abrahamsson [2007] where sprint retrospectives are used for improvements and knowledge transfer. They propose an improved method for Software Process Improvement enabled by Post Iteration Workshops, even though it is not related to interactive research**.**

On six occasions, during the regular Scrum retrospectives, questionnaires were used to collect experiences from the respondents. After a further three months, a final questionnaire was used, followed by individual discussions around the responses. The team was informed about the research project from a methodology perspective and everyone had the option to participate or not in the questions session.

2.3.5 The researcher's role in industrial research

In industrial research when the researcher interacts with practitioners in a development project, it is important to be clear about and reflect over the roles in different situations. It is possible to define three roles that are typically involved in an industrial PhD project and that the PhD candidate may potentially play during periods of the project:

- Practitioner – performs systems and product development with the objective to deliver products or services to downstream groups or to an end customer
- Method engineer – has a focus on introduction (or improvement) of methods and their related processes and tools in the industrial environment in order to increase quality or reduce cost/lead-time. Usually, several factors change during a project lifecycle, imposing a 'continuous change management' where the method engineer should play a central role
- Researcher – has a focus on building knowledge for academia and for industry, to suggest new or modified methods / techniques based on insights from analysis and influences from other research.

As the PhD research was performed part-time with other obligation in the company during the rest of the time, it was essential to keep the researcher, methods engineer, and practitioner roles separate. The separation of roles was simplified by a geographical separation; three different offices/desks were available, one at Linköping University, one at the Saab Simulation-Center, and a third at the Method & Tool department at Saab Aeronautics.

2.3.6 Overview and summary

An overview of the research from a methodology view related to systems development is shown in Figure 6. Each paper is positioned in the systems development V-model, [INCOSE 2010].

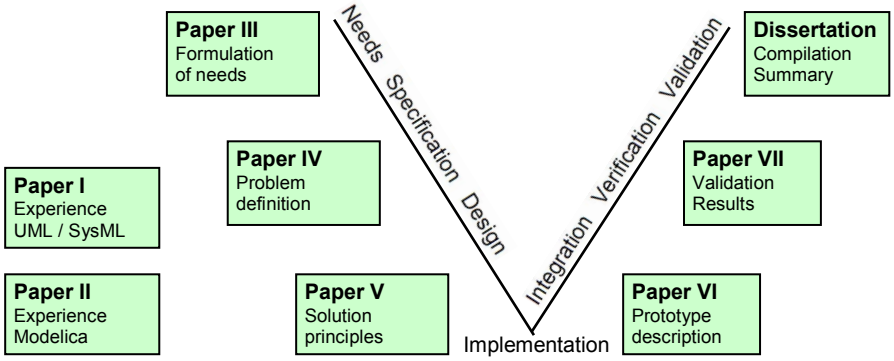

Figure 6. The V-model used to visualize the position of each paper during development of the configurator prototype. Paper [I] and [II] constitutes a part of the background.

To summarize the research methods description; System Development is used as the main research approach, and the last phase of the work is well aligned with "Industry-as-laboratory". The results of the prototype-implementation are validated through Scrum retrospectives.

2.4 Contribution

In summary, this research provides contribution to the interdisciplinary area:

- model based systems engineering, mainly modeling and simulation
- knowledge based engineering and product line management
- product data management / software configuration management

In the first phase, the main contribution is the definition of the Modeling Domain framework that enables classification of modeling methods and tools in large-scale systems and product development, see [Andersson 2009]. The contribution from the second phase is more industry-oriented and provides solutions to enable large-scale model based development and management of simulation models in order to shorten lead-time and improve quality. Contributions are covered in more detail in section 7.3.

2.4.1 Scientific Contribution

The scientific contribution of this work is basic principles for reuse of knowledge and data in the combination of model based and product line engineering. Products are traditionally handled in Product Data Management (PDM) system. Such systems are designed for collaboration between traditional engineering domains. The simulation domain, however, is not yet fully supported. By developing an information model and demonstrating a prototype tool for mapping of simulator information from the PDM to the simulation domain, knowledge is gained about configurability and collaboration effects. One idea is the definition of primary and secondary product lines. This description clarifies industrial needs and constitutes a basis for further research and development in the field of methods to support product line engineering of 'multi-product lines' and complex industrial products.

2.4.2 Industrial Contribution

Many research programs in this area can present powerful techniques and methods with small examples or with a specific or narrow problem to solve, but scaling them up to industrial usage is sometimes of less concern. In this work, scalability is one of the underlying areas of interest, with the assumption that results should be usable for a wider range of employees, not only graduates or specialists. Engineering configurators is foreseen to be a basic capability of future product development as companies strive for product line architectures.

3
Model based development

MODELS OF SYSTEMS and products increase in value as more and more knowledge is kept within models. Models may be of many different kinds, from cost estimation to spare part logistics. The focus in this dissertation is on models representing a system (e.g. an aircraft's fuel or navigation system), that is composed of hardware, software and, where applicable, human interaction, as shown in Figure 7. A complete aircraft model is in turn made up of several such system models, but is still a system model, even though at a higher level of complexity. At an abstract level, these system models define the names and relationships between parts: these are collectively called system architecture or structure. When details of functionality, flows, and physical equations are added, the model can be used to predict performance and dynamic behavior; it becomes a simulation model.

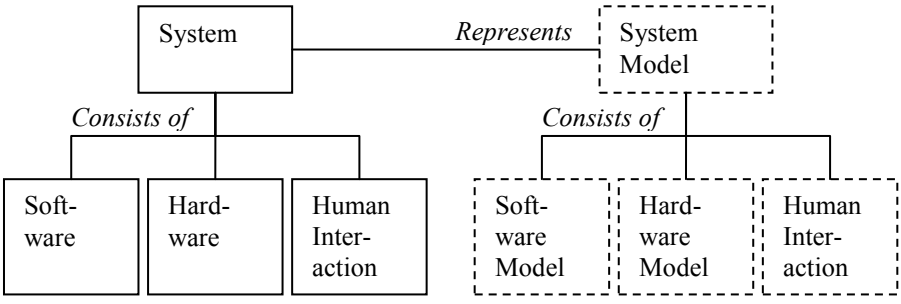

Figure 7. A system model represents a system. The composition of the system model is preferably a replication of the system composition.

3.1 Systems Engineering

In this dissertation, Systems Engineering (SE) is interpreted as the engineering activities that are general regardless of technical discipline. It includes integration of the engineering and project management interface, but also integrates work in the different technical disciplines, as illustrated in Figure 8.

This understanding of SE is mainly based on INCOSE definitions and the INCOSE Systems Engineering Handbook, [INCOSE 2010].

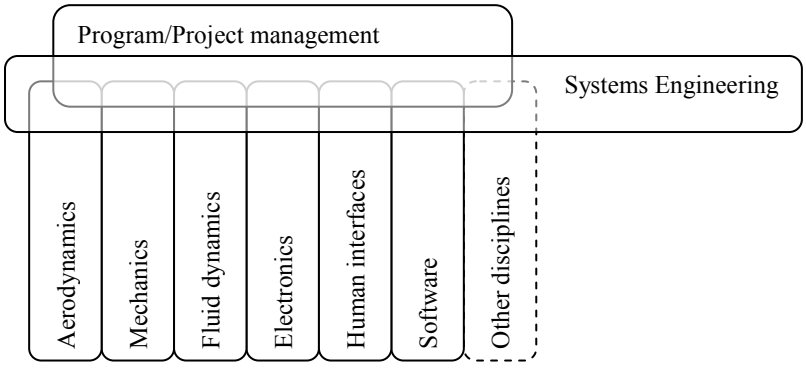

Figure 8. Illustration of systems engineering in relation to other engineering and management disciplines.

Activities included in SE are typically:

- Specification & Requirements Management
- Product breakdown & architecture
- Management of engineering budgets; Weight, Power, Cooling
- Modeling, Simulation & Optimization techniques
- Risk Status and Control
- Subcontractor Management
- "-ilities", e.g. Safety, Availability, Reliability, Maintainability, Reusability
- Planning; Writing the Systems Engineering Management Plan

The planning of engineering methods/activities is most important in a project's start-up phase, but has to be ongoing throughout the project as it includes activities in a product lifecycle perspective, which are not all possible to set at an early point. Here a just-in-time approach is preferable; decisions, descriptions, and education in each respective activity/practice are done just ahead of when they are required in the project.

To be able to plan and perform expansion of the capacity of the various SE tasks requires good understanding and knowledge about the existing organization. A conceptual model that illustrates the overall organizational performance and weaknesses of certain abilities is *the balanced barrel of SE*, se Figure 9.

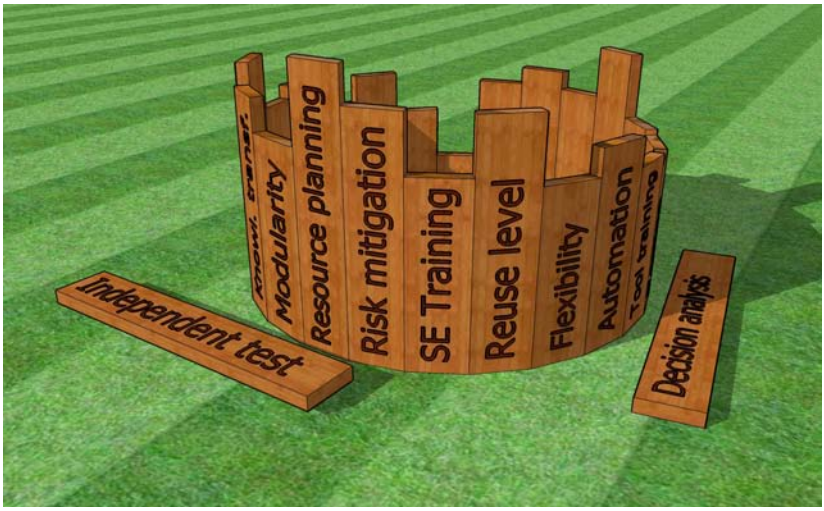

Figure 9. A conceptual model; the balanced barrel of systems engineering. Each segment represents the value of a systems engineering activity. How much wine the barrel may contain represents the overall performance of the organization and is thus dependent on the shortest segment in the barrel. In this example, "independent test" and "decision analysis" are potential activities not yet implemented formally.

A well-documented collection of best practices with the aim of helping organizations to improve their development processes is the CMMI® (Capability Maturity Model Integration), e.g. the *CMMI for Development* [CMMI Product Team 2010]. CMMI has a focus on maturity and balanced introduction of practices in an organization. According to CMMI for example, it is not recommended to invest in *Causal Analysis and Resolution* before *Configuration Management* has been established.

3.2 Development process models and standards

A range of defined models/processes exists for different activities within the industrial development of complex products. Development models used in aerospace are adapted from those and instantiated for specific needs. As the aerospace area is a wide one, further other useful standards, for example in avionics, are adopted from the electronics and communication areas. Interchange of knowledge and standards between the automotive and the aerospace sectors in the area of methods development is in progress. This section introduces some definitions to support systems and product development.

3.2.1 ANSI/EIA-632 - Processes for Engineering a System

The ANSI/EIA-632 [1999] standard "Processes for Engineering a System" from *American National Standards Institute* defines an approach to engineer (or re-engineer) a system, incorporating industry best practices. The approach has three major parts:

a) A system is one or more end products and a set of related enabling products that allow end products to meet stakeholder needs and expectations

b) Products are an integrated composite of hierarchical elements, integrated to meet the defined stakeholder requirements

c) The engineering of a system and its related products is accomplished by applying a set of processes to each element by a team having the needed knowledge/skills.

A system consists generally of a product breakdown and specification structure as described in Figure 10.

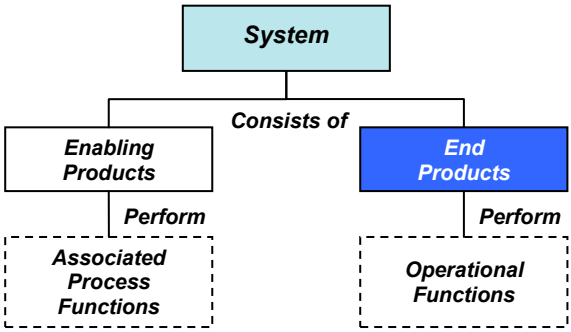

Figure 10. ANSI/EIA-632 definition of Enabling and End products.

Each product is broken into sub-systems in a hierarchical manner shown in Figure 11. This explicitly means that each system at every level has its own set of enabling products, which in the model based case include the actual models of the end product(s).

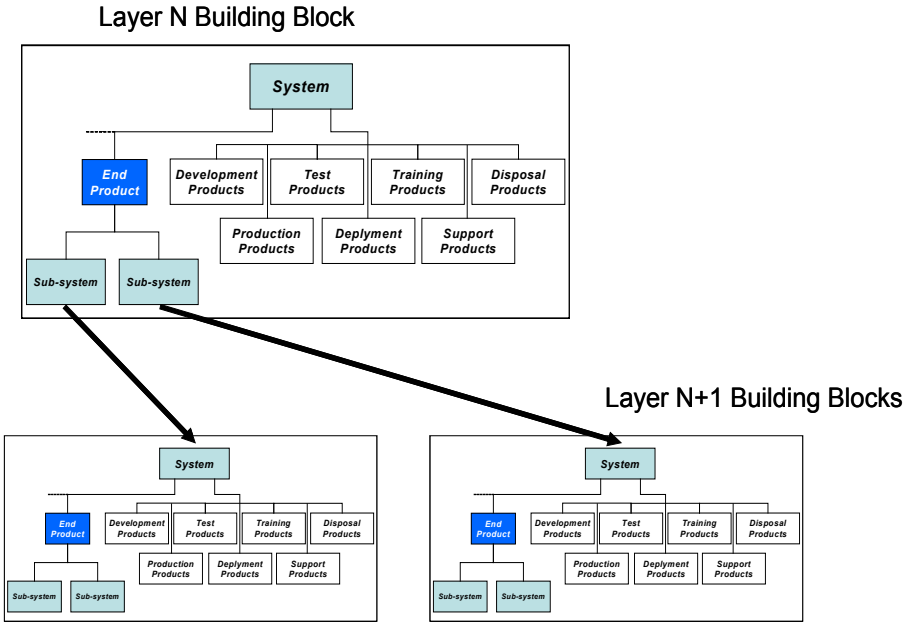

Figure 11. Building blocks in layers according to ANSI/EIA-632.

The ANSI/EIA-632 standard clearly distinguishes between "acquirer requirements" and "other stakeholder requirements". Sources of other stakeholder requirements include government and industry regulations, international conventions, environ-mental constraints, and company directives. In general, other stakeholder requirements place constraints on the system development, both on the resulting product and the processes for developing it. It is usually impossible to meet all requirements for a particular system since they are conflicting relative to one another, so early and thorough requirements analysis is crucial, preferably by means of modeling (and simulation when appropriate).

3.2.2 Product and system lifecycle

A widely used systems/software development model is the two-dimensional model with system lifecycle phases versus process activities according to ISO/IEC 15288 [2008]. It establishes a framework for describing the life cycle of systems by defining a set of processes and associated terminology. In a multi-customer scenario with a product family strategy, the traditional product lifecycle model should be enhanced with a system lifecycle definition that includes the "system-phases" of a whole product family seen from a development point of view. This definition may serve as a template when designing or changing the engineering environment (selection of methods and tools). Each system-phase requires different capabilities and performance of the engineering environment.

Table 2. Definition of system-phases for a product family, from [Andersson 2009].

System phase	**Conception**	**Core development**	**New variants**	**Enhancements**	**Maintenance**
Main work	User needs elicitation Explore existing products Trade-off study Optimization	Definition, specification, design, implementation and initial production	Variant specification and verification. Configuration Production automation	Rework of system, integration of new functions/features Obsolescence management	Maintenance and support of system Corrections User feedback handling
Main objective	Defined scope for products	First product release	Defined product family	Keep products competitive	Keep customers satisfied

The system lifecycle phases in Table 2 are used to analyze the long-term effects of different choices of development method and its supporting engineering environment.

3.2.3 Iterative and incremental methods

Developing and delivering a larger system in increments is a way of reducing risk. Incremental methods for model driven software development have a history dating back to the early 1990s. Since then, a number of software development methods have appeared, ranging from the waterfall method to highly-incremental methods like the extreme programming (XP) method [Beck & Andres 2005]. The Scrum method is similar to XP but has continuous improvement with retrospectives as a prescribed activity as described in section 2.3.4 "Scrum as method to support interactive research" above.

3.2.4 Standardized languages and modeling notations

There are many standardized modeling languages, but also tools with de-facto-standard notations. Below, some languages and modeling notations most relevant for this dissertation will be described.

XML

The XML language, World Wide Web Consortium [2008] is widely used to capture information and meta-data and for transformations. When the information structures are defined, XML schemas (.xls) and transformations (.xslt) are standard features, enabling information exchange across engineering domains/tools. Johansson [2003] shows that simulation models can be specified in the XML format, and then transformed to a domain specific language, for example Modelica. Examples of XML-based standardization for use in modeling, and exchange of product data are described below.

1) Interface Specification

In the area of model interface definition of modular architectures, the Functional Mock-up Interface, FMI, standard provides specifications in XML format; see [Modelisar 2011]. This gives a tool to support integration in different frameworks. A model created with a tool supporting FMI is thereby integrated easily in simulation environment with respect to the signal interface definition. The FMI standard also provides a newly released PDM interface in order to handle "modeling, simulation, and validation information" in the PDM systems.

2) Product data

Another defined XML-based format for exchange of information for Product Data Management (PDM) is the PLMXML format, which is "*a collaboration/interoperability protocol designed to exchange pertinent PLM information*" between data sources; see further [Siemens 2011].

Modelica

Modelica [Modelica Association 2011] is a descriptive, object-oriented modeling language suited to physically based simulations and analysis of behavior and performance. There are model libraries, both open source and commercial, typically supporting mechanical, electrical, electronic, hydraulic, thermal, electric power or process-oriented components. An example of an ideal capacitor component is presented in the textual model definition below.

```
model Capacitor
  Pin p;
  Pin n;
  Real v;
  Real i;
  parameter Real C "Capacitance";
equation
  0 = C * der(v) - i;
  0 = p.v - n.v - v;
  0 = p.i + n.i;
  0 = p.i - i;
end Capacitor;
```

In this model, the parameter (C) is used to instantiate the model with different values of its capacitance. This kind of parameter-sized component is well suited for reuse and sharing between users through a model library. Experience from industrial usage of Modelica in simulation of aircraft vehicle systems has been published in paper [II]. As reference to modeling with Modelica, see [Fritzson 2004].

Unified Modeling Language and Systems Modeling Language

The standardization organization Object Management Group (OMG) has released specifications for the Unified Modeling Language (UML®) [Object Management Group 2007], and the Systems Modeling Language (SysML™) [Object Management Group 2008]. Both are general-purpose object-oriented graphical modeling languages for specifying, analyzing, designing, and verifying complex systems. UML provides graphical notation with a semantic foundation for modeling behavior and structure. SysML represents a subset of UML with extensions for requirements and parametrics (basic mathematical support) needed for Systems Engineering. Both have weak support for building simulation models, but most tools that support UML/SysML, have code generation engines, enabling compilation and execution. The defined SysML diagram types are shown in Figure 12.

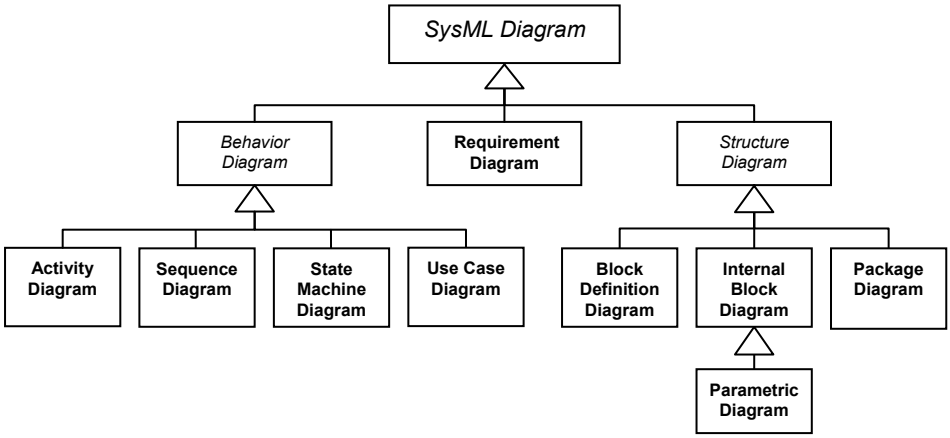

Figure 12. Diagram types defined in SysML 1.1.

It is convenient for a specific project to reduce the set of UML/SysML diagrams used, as there is some overlap between the diagram types. A limited set also simplifies the introduction of UML/SysML modeling including guidelines, training, and tool set-up. As described in Paper [I], an appropriate set of diagrams to use in avionics system development is:

- Use Case and Activity Diagrams for analysis.
- Class/Block Definition, Sequence, State Machine, and Deployment Diagrams for design, implementation, and test.

SysML also has built-in definitions of dimensions and SI-units (e.g. "**`dimension; Frequency, unit; Hertz`**" or "**`dimension; Power, unit; Watt`**"). It is possible to add "user-defined" units, which is powerful in avionics/aviation specifica-

tion and design. Handling of, for example, flight speed and altitude is, in the aviation community, done in the non-SI-units **`Knot`** and **`Foot`**. For further details on the SysML language and examples of its use, see [Herzog 2005; Weilkiens 2008; Friedenthal, Moore & Steiner 2011].

Experience at Saab Aeronautics from introduction of UML/SysML as a means for systems modeling in a large organization supported by the IBM® Rational® Rhapsody® tool [IBM Rational Rhapsody 2011] has been reported in paper [I] and in Herzog, Andersson, and Hallonquist [2010].

3.2.5 Model driven architecture

The Model Driven Architecture (MDA) approach, as described in Mellor and Balcer [2002], is a way to support separation of functional specification from implementation. MDA is used in the development of software intensive systems where automatic code generation is part of the process. Its underlying concept is to separate 'do the right thing' from 'do the thing right' by introducing platform-independent models (PIMs) and platform-specific models (PSMs). Translation from PIM to PSM is defined by rules in a platform definition model and generally performed by automated tools. Translation (or generation) from models to different source code languages, such as ADA, C++ or Java is used, but also translation to documentation of the design.

3.3 Classification of models and modeling domains

Ever since modeling became a practice for specification or problem solving in science and engineering, the number of available techniques and tools has increased. This is partly caused by the evolution of work stations/computers, but also thanks to the demonstrated value of modeling in the area of complex problems. Naturally, every modeling technique fits best for one small set of "problems", even though it may be used for a broader set. In large development projects, it comes to a choice or trade-off between on one hand the use of many specialized, powerful tools, and on the other hand the use of a few multipurpose, but usually "dull", tools and techniques. Many attempts have been made to classify modeling techniques, and some classifications are mentioned herein.

3.3.1 Value and acceptance of models

When choosing a modeling technique, it should plausibly add sufficient value to the project, and it is important to recognize the purpose the modeling has and what characteristics the different techniques have. One fundamental purpose of M&S is to reduce the amount of physical prototyping and testing – activities which normally demand a significant amount of resources. Related aims are to enhance the abilities to take early model based design decisions, and to use M&S to support the certification of aircraft systems, [CRESCENDO 2010]. In order to achieve this, one should be able to answer questions like; *To what extent can we trust the model, how well does the model represent the real system, do we know the limits of the model, does the model cover the intended use?*

The above questions deal with M&S credibility. Depending on the model's complexity and the intended use, performing a relevant assessment of a model's credibility might

be a challenging task. Research is being done in this field, where different methods are proposed for making such an assessment. Common ingredients of methods are verification and validation aspects. Several definitions of these terms exist and one mature reference for definition of verification and validation is from NASA's Standard for Models and Simulations [NASA 2008]:

Verification: *The process of determining that a computational model accurately represents the underlying mathematical model and its solution from the perspective of the intended uses of M&S.*

Validation: *The process of determining the degree to which a model or a simulation is an accurate representation of the real world from the perspective of the intended uses of the model or the simulation.*

Another common aspect is related to M&S uncertainty management, which here refers to the process of identifying, quantifying, and assessing the impact of sources of uncertainty embedded along the development and usage of simulation models. Some potential sources of uncertainty are model parameters, model input data, model simplifications, and the numerical method used by the solver. Several definitions aimed to distinguish between different types of uncertainties are found in literature, [Oberkampf, DeLand, Rutherford, Diegert & Alvin 2002; Thunnissen 2005; Padulo 2009]. Application for aircraft vehicle systems is found in [Steinkellner 2011]. Commonly, a distinction is made between *aleatory uncertainty* (due to statistical variations, also referred to as variability, inherent uncertainty, irreducible uncertainty, or stochastic uncertainty) and *epistemic uncertainty* (due to lack of information, also referred to as reducible or subjective uncertainty).

For a model to add value in the end, it needs to be accepted for use, regardless of uncertainties or known limitations. An emerging standard for V&V and Acceptance is the Generic Methodology for Verification and Validation (GM-VV) [GM-VV 2010]. It provides a handbook, which guides its users through the V&V and Acceptance efforts and clarifies their responsibilities and how to apply the methodology in practice. It also describes how to tailor the methodology to the needs of a specific M&S project.

3.3.2 Specification and Analysis models

One classification is to divide models into "specification" and "analysis" models.

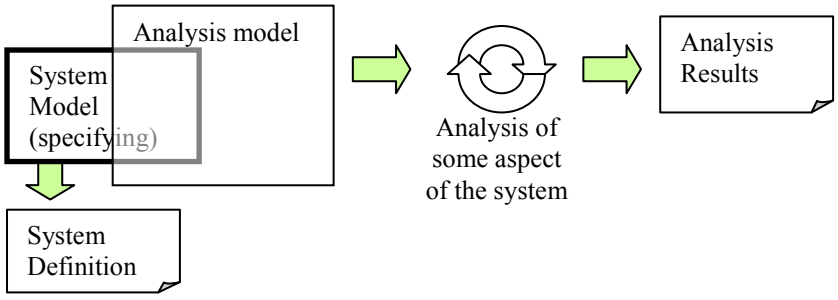

Figure 13. A specifying model is the basis for definition and analyses of a system.

An example from solid modeling and hardware/structure development is the following:

- A specification model is the definition of surfaces (shape) and the content (materials) of a component. It is typically done in a 3D CAD (Computer-aided design) tool, in a visual prototyping manner.
- A connected analysis model is used for stress analysis on the same component, based on the specification, but with information on boundary conditions (spectrum of forces) added.

An analysis is performed with a subset of information from the specification model, but with additional information for the specific analysis to be performed, as shown in Figure 13. The same specification system model can consequently be the basis for performing analyses of several aspects of the system. Examples of analysis from avionics design are fault tree analysis (TFA), formal methods analysis, and analysis by simulation.

3.3.3 Modeling domains

Modeling domains are a framework for classification of modeling tools and their related techniques/methods. Classifying and sorting tools/techniques/methods has been a means to analyze strengths and weaknesses of different modeling methods/tools and to organize the work in business improvement programs and in engineering process research.

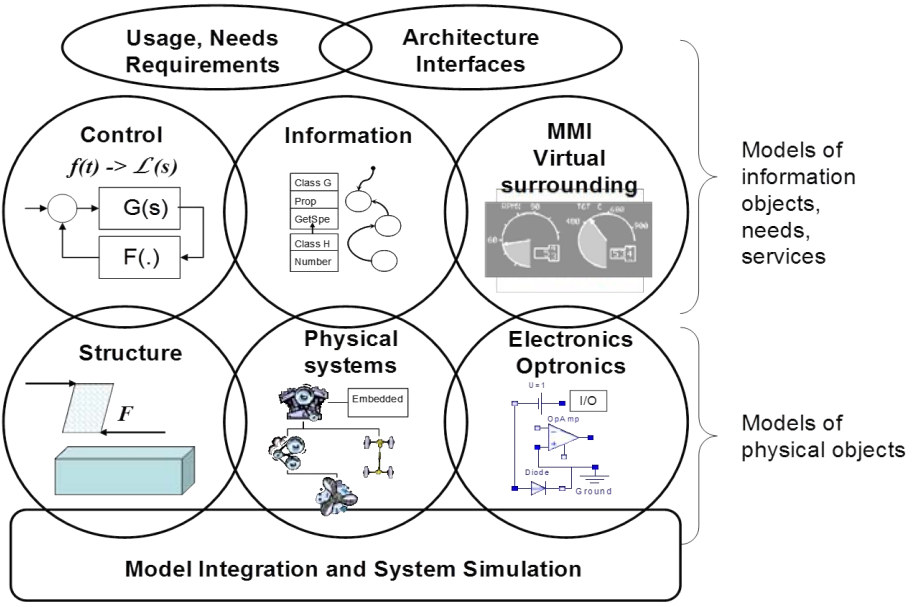

Figure 14. Definition of Modeling Domains. The lower part is related to physical objects and their properties, such as; space, time, energy and matter, whereas the upper part relates to information.

The framework is for example used within the CRESCENDO research project [CRESCENDO 2010] and by the Process, Methods & Tools (PM&T) organization at Saab Aeronautics. When model based techniques are introduced, the framework aims to analyze whether the change efforts are broad enough and whether all domains are cov-

ered by appropriate enhancements to achieve engineering efficiency, quality, and an attractive engineering environment. The modeling domains are defined as shown in Figure 14. A further description of each modeling domain is found in Andersson [2009], which is one basis for this dissertation. Most of the work in the remaining chapters of the dissertation is related to the "Model Integration and System Simulation" domain.

3.3.4 Behavioral modeling techniques

In order to create a description of behavior, a number of modeling elements are required. The necessary set of semantic elements, as defined in [Oliver, Kelliher & Keegan 1997], includes:

- functions, which accept inputs and transform them into outputs
- inputs and outputs, of various types, and
- control operators, which define the ordering of functions

One example of classification of behavioral models is found in Cassandras and Lafortune [2008] that holds for Discrete Event Systems (DES) modeling techniques. Examples of different types are event-driven, discrete-state, non-linear, time-invariant, and dynamic models. For modeling of physical systems, other classifications are used, for example continuous/discrete time; see further Andersson [2009] and Steinkellner [2011].

3.4 Simulation of complex products

The main objective for simulation in aerospace is to reduce risk and cost. In the early stages, risk and cost are reduced by gaining a better understanding of how to specify the system/product and what the tough constraints are. In later stages, simulation generates "flight-hours" providing engineers, pilots, and other stakeholders with knowledge about the system for different purposes.

3.4.1 Modeling and simulation environments

Today, there are many advanced domain modeling and simulation environments that allow detailed simulation prior to components' and products' realization. Those environments evolve continuously regarding both languages and modeling techniques. Languages/tools such as Mathworks, Simulink® [2011], Modelica® [Modelica Association 2011], UML® [Object Management Group 2007], and VHDL-AMS Christen and Bakalar [1999] are increasingly used in industry.

For a heterogeneous system such as an aircraft, there is a need to combine and integrate simulation models developed in different environments into a virtual system in order to simulate the complete system. *Closed-loop-simulation* denotes an arrangement of connected models in a loop, including sensors and actuators, for example an ECS model connected to its control logic. Closed-loop-simulation is further described in papers [II] and [V].

Simulations are used to predict the behavior and performance of system configurations not yet realized, but also of real product configurations for verification activities. Data from the real system is fed back to the models and to the language/tool-specific block-libraries to improve the accuracy and quality of simulations. Different integration tech-

niques for simulation models, and different types and scales of simulations can be identified as described in the following sub-chapters.

3.4.2 Integration strategy

Information and models from different integration levels and from different modeling domains often have to be integrated to be part of analysis and/or verification activities. Carloni et al. [2006] argue that a single environment cannot offer a complete solution to the needs of designers who use hybrid/integrated models to represent the system under development.

In Saab Aeronautics' experience, a "loose integration" strategy should be chosen in order to avoid lock-in effects and costly long-term maintenance of the tool integrations. This requires formats and interfaces to be clearly defined or standardized in order to integrate the tools and maintain this integration. Tight integration, on the other hand, relies on the tools being connected and running in parallel with exchange of data when performing an analysis or a simulation. The loose integration strategy reduces the dependency on two or several tools being available simultaneously.

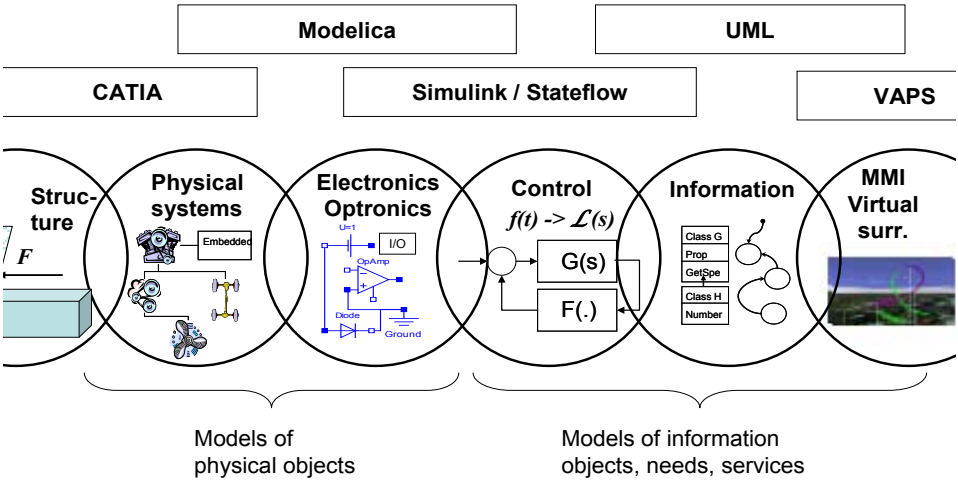

Figure 15. Integration of modeling domains and examples of tools.

There are several interfaces between the modeling domains that need to be integrated. In Figure 15, the main pattern of integration needs is shown, going from a "heavy" hardware/structure to the left, through equipment, electronics, control, and information, to a "soft" graphical layout to the right. When analyzing features of development tools it is interesting to notice that many tool vendors try to cover larger and larger parts of this domain map by adding features and functions to their tools. Three examples, with reference to Figure 15, are:

- Modelica is integrated into the new releases of CATIA (CATIA Systems Dynamic Behavior), which extends its capabilities for dynamic simulation (extension of CATIA to the right in the figure); see [Dassault 2011].

- In the Modelica® language, libraries have been developed for Petri nets and state charts to extend Modelica (to the right in the figure); see [Modelica Association 2011].
- Simulink® is integrated with Stateflow® (extension to the right in the figure) and enhanced with Simscape™ (extension to the left in the figure), see Mathworks, Simulink [2011], Mathworks, Stateflow [2011], and Mathworks, Simscape [2011].

Tools with functionality that supports multiple modeling domains are of course an advantage; they give the engineers and the team a possibility to choose from several tools for a specific engineering task. In large-scale projects, however, it is a balance between uniformity and diversity in the set of supported modeling techniques. With several teams established over a period, each one focused on specific engineering tasks, it might give an overall diversity of tools and ways they are used. This may lead to a sub-optimized implementation of methods/tools and inefficiency in the long term, especially regarding specification modeling.

Concerning modeling for simulation, connection (or integration) of the modeling domains is done at different levels. At component level, integrated simulation can be done using co-simulation or hosted simulation techniques, performed in desktop tools. At higher levels of integration, more execution-efficient techniques appropriate for large-scale simulation have to be used.

3.4.3 Integration techniques for simulation models

Naturally, simulation models are developed relying on different modeling techniques – each focused on a specific problem, engineering discipline, or aircraft subsystem. From the aircraft integration perspective, different models need to be integrated into a larger model for simulation/analysis with a broader scope or at a higher system level.

It is a growing challenge to use and integrate simulation models from different domains, as an increasing part of the end system verification relies on results from simulation models rather than expensive testing in flight tests. The development of computer performance and modeling-and-simulation tools enables simulation on larger and larger scales. Consequently, the need for integrated models and their validation is growing. Simulation (sub-)models for aircraft systems can be organized into the following major categories:

- Equipment models (e.g. resistors and capacitors in electronic systems, pipes and nozzles in hydraulics) for performance evaluation and dimensioning;
- Models of the embedded software for control of system functions and monitoring of functions and of the equipment/hardware;
- Models of the environment of the system.

Several commercial and in-house developed tools exist for connecting different simulation models and there are different ways of performing integrated simulation.

Co-simulation

The concept of Co-simulation is to connect and integrate models combined from different simulation engines and execute them concurrently. This approach is supported by tool vendors who provide add-ons or packages for connecting tools together for simula-

tion set-up. Co-simulation is suitable for a system of Electronic Control Units (ECU) connected by a data bus, but less useful when the connected models have physical interactions, requiring power port modeling techniques and a specific equation solver. In co-simulation, the tools are connected [Öström, Lähteenmäki & Viitanen 2008].

Hosted Simulation

In Hosted Simulation, the simulation engine from one of the modeling and simulation tools is used to "host" other models during simulation. The hosted simulation method is enabled through code generation. A model created in one tool is simply generated to executable code and imported (hosted) in another tool to perform the simulation. In hosted simulation the codes are connected. A comparison of two variants (different hosting tools) of hosted simulation can be found in Steinkellner, Andersson, Krus and Lind [2008].

High Level Architecture

High Level Architecture (HLA) is a general-purpose architecture and standard for connecting simulation applications. HLA defines an architecture with a set of Application Programmer's Interface (API) standards. Simulation applications communicate by making calls to the HLA APIs. HLA can be used for integration of several distributed simulators by network connection over long distances. Parts of a simulator may also be connected through HLA, for example an *aircraft simulation model* and the *tactical environment*.

3.4.4 From desktop simulation to the iron bird

Tests can be performed in software-based simulators and/or in hardware based system simulators (rigs) with product-equivalent computers and other equipment in the loop. There are different types of simulation facilities:

- Desktop simulation tools, cheap and easy to access
- Handling qualities, software based, simulator with or without pilot-in-the-loop
- Presentation and maneuvering simulator with Human-Machine-Interaction in focus
- System simulator (rig) with a large amount of target hardware and other product-equivalent equipment present. This type of simulation is defined as large-scale.

A picture of a simpler kind of simulator for mid-scale human-in-the-loop simulations is shown in Figure 16.

Figure 16. A simple human-in-the-loop simulator used for validation and verification of system functionality, but also for basic training of engineers.

Access to a simulator with actual system status is valuable to the engineering teams in order to gain a common understanding of both implemented system functionality and remaining work.

For test and verification of safety critical functions prior to first flight of a new aircraft configuration, the classical iron bird has long been used. An iron bird with complementary simulations in various domain-specific models/tools is the predominant method of collecting evidence for verification/certification prior to flight. The trend is however to increasingly rely on software based simulations, see [II]. The notion of "virtual iron bird" is used to describe a model used for analysis that earlier had to rely on a classic physical rig. Schallert et al. show the benefits of using a virtual iron bird with the possibility to perform optimizations by means of simulations to evaluate power consumption of different alternatives with the aim of minimizing the generator power to be installed on a future all-electric aircraft [Schallert, Pfeiffer & Bals 2006].

3.4.5 Mid-scale simulation

Here, mid-scale simulation is defined as the activity performed, when some simulation models of aircraft subsystems, developed with different modeling techniques, are integrated into a larger model, complex enough to not be simulatable in one desktop modeling and simulation tool.

A software-based simulation model of this scale is easy to execute with a range of different user scenarios, and this can be done in "batch mode", preferably distributed during "non-work-hours", to maximize the usage of the company's computational recourses. A set of scenarios are created with selected values of inputs (e.g. load configuration, fuel content, speed, and altitude). Out of several thousand simulated scenarios/maneuvers, including degraded modes, there is a selection of critical maneuvers for further verification. For each payload combination, a set of the most severe and critical maneuvers are selected for pilot-in-the-loop simulations and maybe flight tests. Not all simulation environments have full support for simulations in batch mode. In paper [II], the lack of support, e.g. for data processing, during batch simulations in a Modelica environment is described.

One drawback of using only software models is that some aspects are difficult to cover. For example, when verifying multi-channel systems, the inter-channel behavior such as redundancy policy, timing, and performance aspects has to be tested separately, in a hardware-based rig with target software, in parallel with the model based functional verification. Therefore, test rigs with the system's real hardware components have to be built to cover those aspects.

3.4.6 Large-scale simulation

When several simulation models of the aircraft subsystems are integrated and specific arrangements for performance or interoperability exist, the simulation is considered large-scale. Examples of such arrangements are real-time execution including pilot-in-the-loop simulation, see Figure 17, or hardware in-the-loop simulation (HILS) configurations.

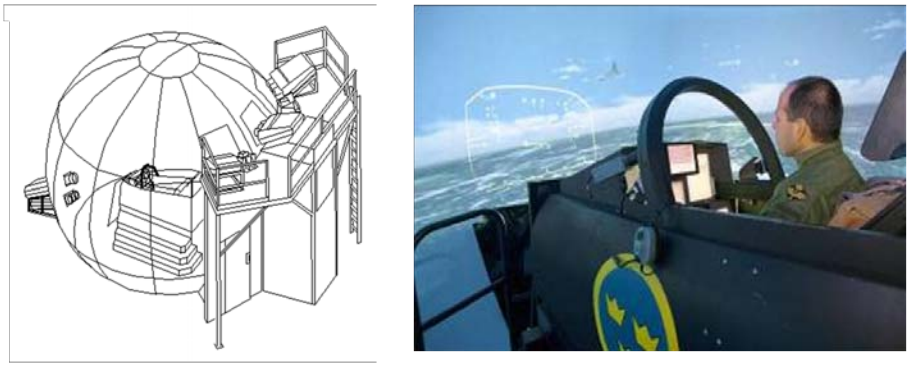

Figure 17. Example of a large-scale simulator with advanced pilot interaction. This kind of simulator facilities also needs a separate Instructor Operating Station (IOS) for setting up scenarios and providing the system and the pilot with e.g. faults and mishaps.

Simulators in the aerospace sector have evolved towards virtual reality machines, and they are sharing some technology with products found in the computer game industry, e.g. visualization technology [Stone, Panfilov & Shukshunov 2011].

A defined product structure of a simulator enables well-described parts including their interfaces, which is the basis to declare a simulator test-worthy, and declaration of design and performance to customers.

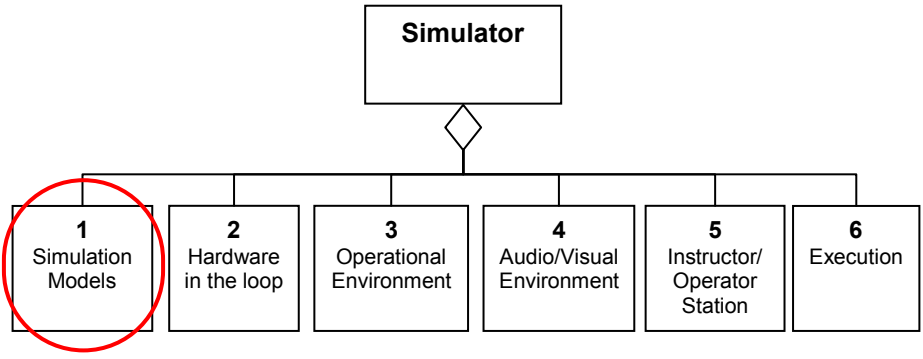

Figure 18. The top level of a generic product structure for large-scale simulators. Part 1-*Simulation Models* is the focus for configuration support related to structures and data of the simulated product.

Figure 18 shows the generic parts (subsystems) of a simulator. Descriptions of the subsystems and some of their functional responsibilities are listed here:

1. **Simulation Models** representing the simulated system/product, for instance an aircraft with its immediate surroundings (including ambient temperature, air pressure, and aerodynamic forces). This part contains several sub-models, which are needed for the simulation of a complex product. Some of the models may be reus-

able and/or configurable for multiple purposes. It includes the connection of models, parameter libraries, mathematics library functions, and solvers needed for calculation and execution during the simulation.

2. **Hardware in the loop** contains functions that enable the connection of vehicle operational hardware (ECUs) so that they can be part of the HILS – hardware in-the-loop simulation.
   - Connection between a/c computer hardware and simulation computer
   - Power supply and cooling air supply
3. **Operational Environment** performs simulation of other vehicles/platforms/systems that interact with the simulated system/product. In military applications, it is called 'tactical environment' and the entities may be hostile (foes) or friendly (friends).
4. **Audio/Visual Environment** (out the window & cueing) creates visualization of the outside world and sensor images as well as presentation of the environment.
   - Generation of sensor data
   - Generation of sound/audio
5. **Instructor/Operator Station (IOS)** & Other Tools has the human-machine interface for control of the simulator including tactical simulation.
   - Simulation control and monitoring (start, stop, load, etc.)
   - Controls for registration and evaluation
   - Debugging functions
   - Control of tactical simulation
6. **Execution** contains hardware and software components for execution. If required it includes real-time management, for simulator in real-time.
   - Simulation computers with operating system
   - Synchronization, data distribution, and recording functions

Of the six parts, the Simulation Models (part 1 in Figure 18) have a strong relation to the primary product (the aircraft).

3.5 Simulation of what-if scenarios

Objectives for simulation during development of aerospace products include reducing risk and cost. System safety is an important aspect of the development. Data from simulations and from measurements need to be available for safety analysis so that the products can be certified for flight and for operational usage. In order to provide data and analysis results to the certification activity, many different "what-if" scenarios are defined. Typical scenarios used for safety analysis are behavior and performance during malfunctioning or errors in the simulated system. Actually, most effort/time spent in simulators for aerospace systems is related to "misbehavior" of the system. This applies for all the contexts: development, verification, and training. For example, training in

landing routines with degraded fuel and/or ECS system functions is important for both pilots and ground crew.

> *"The deployment of highly skilled staff is an essential prerequisite for the safe and effective operation of aerospace systems. Simulation-based training plays an increasingly important role in the qualification of aerospace systems personnel."* [Stone et al. 2011]

Simulation-based training of pilots and other personnel in the aerospace business is certainly important. However, there are signs of overconfidence in its effect and how much of the flight training that should be performed in simulated environments. This is a quote from a patent on a flight simulator layout for pilot training:

> *"Due to recent advances in simulator technology, simulators are expected to supplant all in-flight training of pilots, and therefore, commercial pilots can expect to receive their Federal License directly upon the completion of their flight training in simulators. Such simulator-trained commercial pilots, then, could carry paying passengers, even though they have never flow an actual aircraft before."* [Geiger 1982]

Model support is needed for many different kinds of fault conditions that need to be simulated and analyzed, or for which training should be performed. What faults to model is, of course, dependent on the system concerned, but a rule of thumb is that sensors and actuators in a system are error-prone. How to model faulty conditions of a sensor is important and there are many different ways to do it; a few examples are listed below:

- Intermittent fault is a fault that repeatedly occurs and disappears. Example: loose connectors
- Incipient fault is a fault that gradually develops from no fault to a larger and larger one. Example: a slow degradation of a component
- Abrupt change is a fault that appears as a very quick change of a variable. Example: sudden breakdown of a component.

For general handling of faults in a sensor or in its connection, it is convenient to build a fault injection feature into the modeling framework. All sensor models are then enhanced with extra output settings, see Figure 19.

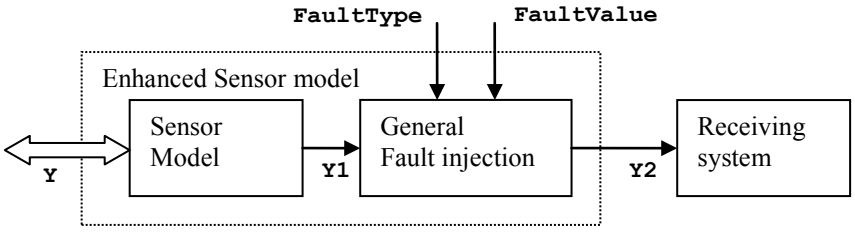

Figure 19. Sensor model enhanced with general fault injection feature

Here is an example of general fault injection settings of sensor signals used at Saab Aeronautics;

Type	Name	Meaning
0	no fault	`Y2 = Y1;`
1	zero	`Y2 = 0;`
2	+ hard over	`Y2 = + BigNumber;`
3	- hard over	`Y2 = - BigNumber;`
4	bias	`Y2 = Y1 + FaultValue;`
5	gain	`Y2 = Y1 * FaultValue;`
6	user input	`Y2 = FaultValue;`

This kind of general functionality has in Saab Aeronautics' experience proven useful in mid-scale and large-scale modeling and simulation, as it is easy to implement and use. Specific sensor faults are of course needed for certain kinds of analysis, or pilot training, and these are preferably built into the sensor model provided by the equipment/sensor supplier.

3.6 Summary of model based development

Models should contribute to manage complexity in an engineering organization. Model based principles covered in this chapter include specification models aimed at defining a system and analysis models based on the definitions, but enhanced with details to, for example, enable simulation. Modeling domains are defined and exemplified with industry-standard languages and notations such as SysML, UML, XML, and Modelica. Different types of aerospace simulations have different purposes; verification of safety prior to first flight and training for increased skills of aerospace operational personnel.

Challenges in the further development of model / simulation based methods are identified. Models become possible to use in multiple contexts and performance of computers increase, which contributes to an increasing number of models. To be useful, the models need to be verified & validated for each context. It should always be remembered that the simulation models are there to represent some reality, and they can gain a high acceptance only if they show compliance with the represented reality.

4
Reuse and its application

THIS CHAPTER defines a set of principles for reuse in the context of product families and use of knowledge-based engineering. Concepts like Product Line Engineering and Software Product Lines are covered and the chapter is mainly focused on the reuse of simulation models and artifacts in software-intensive systems. Key concepts such as *constraints*, *feature model*, and *Configuration Tasks* are also defined. These concepts and methods are essential to create methods and mechanisms for customization of simulation models, and especially for configurable models that have several purposes- *multi-purpose models*. Analyses of changes in a product line with the help of simulations are also covered.

4.1 Reuse principles

Reuse of existing solutions, components, knowledge, and data/information is not only tempting; it is in many cases wise because it saves time, reduces risks, and there exist earlier experience of using the actual artifact. Reuse of artifacts that are produced in industrial development is a strategy with great potential, not only to reduce lead-time and cost, but also for increasing product quality and job satisfaction [Clements & Northrop 2002].

Reuse can be done in several ways. The prevalent and easy-to-adopt method of reuse, which we have all practiced at some point, is "copy-and-paste", see Figure 20, which in software engineering also is called duplication [Mann 2006].

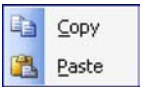

Figure 20. Copy-and-paste tools

Generally, copy-and-paste of engineering artifacts could be done at different levels; source-code line, model file, directory, database, or a complete product, but it has several disadvantages:

- Generation of many copies drives data storage volume requirements

- Problems with ”information divergence”, i.e. the copies will differ with time
- If these copies are subject to controlled change the result will be increasing costs for updates and maintenance
- It is usually a tedious task to update the same data within different copies

Most engineering methods and information management systems have as their basic principle avoiding unnecessary duplication of information. In software engineering, the principle ‘Don't Repeat Yourself’ (DRY) was formulated by Hunt and Thomas [2000], which aimed to reduce repetition of information. The DRY principle is stated as:

> *"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."*

This is applicable not only to software, but also to artifacts like requirements, database schemas, test plans, build systems, and documentation.

Methods to avoid duplication that are more efficient in the longer term need an infrastructure and tools to mechanize the reuse and are based on a *deliberate design choice*. Examples of techniques include:

- Configuration by means of filters and views
- Databases and spreadsheets with report generators, filters and views
- Configuration of a system via parameters
- Configuration/customization through the product line approach.

These techniques do not have all the disadvantages of copy-and-paste, but implementation and maintenance of such techniques require investment in the form of infrastructure, tools, procedures, and training. The method of reuse further covered in this work is based on a design choice utilizing single source and the product line approach as the guiding principles.

4.2 Product Line approach

This section discusses the theories and basic concepts of Product Line Engineering and Software Product Lines used in this dissertation. The basic principles for reuse, product platforms, and Software Product Lines (SPL) are well established, and are in this work mainly based on Clements and Northrop [2002], van der Linden et al. [2007], and Simpson et al. [2006] and the context is configurable simulation models intended for multiple purposes and simulator products.

There exist many definitions of product line, family, and platform in the literature. In this work, *product line* and *product family* are equivalent. *Product line* is mainly used in the combination with the terms architecture, approach, engineering and similar.

4.2.1 Knowledge Based Engineering

Knowledge Based Engineering (KBE) focuses on the reuse of technical knowledge by automation of repeatable, error-prone, and/or time-consuming engineering tasks using tools. In the MOKA methodology (Methodology and Tools Oriented to Knowledge-Based Engineering Applications), KBE is defined as:

> *"The use of advanced software techniques to capture and re-use product and process knowledge in an integrated way"* [MOKA Consortium 2001]

Historically, KBE has its roots in Computer-Aided Design (CAD) [Wikipedia, KBE], and now there are several engineering methods included in the scope of KBE. In the book Intelligent Systems by Hopgood [2001], KBE and the related field Computational Intelligence include engineering methods and techniques such as:

- Design optimization, to find the best design solution
- Handling of uncertainties and supporting methods e.g. Fuzzy Logic
- Expert systems and Neural networks
- Parametric modeling and generation of design configurations
- Rule based systems, constraint programming, and configurators

Searching within a design space is a key in many practical problem-solving tasks. Many activities are aimed at collecting or providing data for searches, comparisons, or calculations.

4.2.2 Platform, variants, and modularity

In product line engineering, similar products are thought of as evolving families that are derived from a common *platform* but with specific, distinguishing features/functions of their own. Each individual product in a product family is called an *instance* or Product Variant (PV). The PVs share common structures and product technologies. An established definition of product platform suitable for the application of industrial products, for instance aircraft and simulators is:

> *"**product platform** - A set of platform elements and architectural rules that enable a group of planned product offerings. Key characteristics of a product platform include:*
>
> *(1) Architectural rules/standards governing how technologies and subsystems ("platform elements") can be integrated;*
>
> *(2) Defines the basic value proposition, competitive differentiation, capabilities, cost structure, and life cycle of a set of product offerings; and*
>
> *(3) Supports multiple product offerings from a single platform, permitting increased leverage and reuse across the product line."* [Simpson et al. 2006]

A product platform may include components (or subsystems; platform elements), processes, knowledge, and relationships. There are three important aspects of a product platform:

- Its modular architecture
- Internal and external interface
- The standards and design rules to which the modules/assets must conform

In platform-based product development, interface definition and management is a central activity. A *modular platform* is used to create Product Variants through the configuration of existing components. Product architecture can be defined as the way in which the functions are arranged into modules and the way in which these modules interact [Ulrich & Eppinger 2008].

For software intensive systems, a slightly different focus is applied because it is possible to create the software products through a software-build process integrated with the development environment for the components. In practice, a software product can from a specification be assembled, compiled, tested, and delivered by automation in a series of integrated steps.

> *"A **software product line** is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"* [Clements & Northrop 2002]

The term *core assets* in this definition, includes reusable software components, requirements, test cases, documentation, models, and even budgets and schedules. The 'core assets' definition is close to the meaning of platform. Key activities in development of a software product line are:

- Core asset development (for example a specification, a software library with mathematic functions, a simulation model). It is important to plan the development of the core assets to be well suited for reuse.
- Product development (specification, building, integration, and delivery of products). The success of this activity depends upon the core assets.
- Management (planning for updates, delivery, resource allocation, etc.)

The key activities are interconnected and *configuration management support* is a central activity for a product line organization and for coordination of the key activities.

4.2.3 Commonality

Degree of commonality is an important property in the analysis and planning of a product family. If many products use many common assets, the reuse potential is large. Different degrees of commonality may be visualized as in Figure 21.

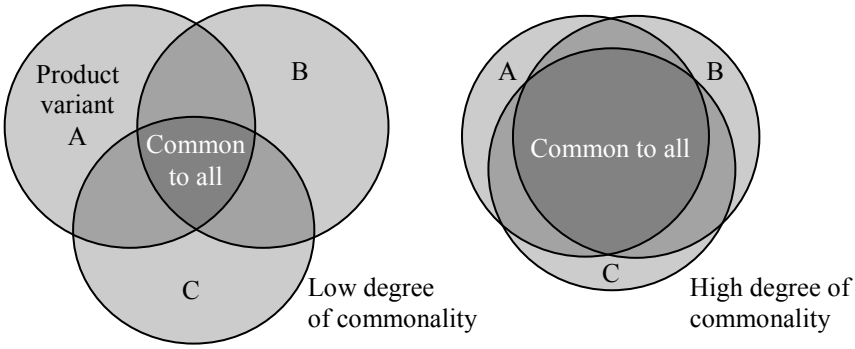

Figure 21. Visualization of degree of commonality, from Sivard [2001].

There is a tradeoff in the degree of commonality; a high degree gives all the benefits of cost saving due to reuse between products, but the cost of coordinating changes, releases, and interconnected product planning, for example, will increase. Wickenberg [2011] describes eight challenges when implementing a product platform in the automotive

industry. One challenge, for example, is when a component matching the toughest requirements, it would typically be over-specified for most uses and hence unnecessary costly. An example is the reuse of an engine in a vehicle product family where the (one size) engine is heavier than the optimal for the smallest vehicle size and too weak for large vehicles. A high degree of commonality might also result in suboptimal products or less distinctive products [Thevenot 2006]. In the literature there are many indices defined to measure the degree of modularity within a product family, see Thevenot [2006] for a summary. A commonality index is typically based on parameters such as the number of common components or the component manufacturing volume.

4.2.4 Variability and variation points

An explicit model of the variability in products and in the core assets is needed for scoping and analysis and as input for implementing configurator support. The variability model defines the variability of different classes of the defined products [Hvam et al. 2008]. It defines what can vary by defining *variation points* for components and types of variation available for a specific variation point, for example their binding time properties. The model also defines variability dependencies and variability constraints to be considered when customizing end products.

The variability model is part of the Product Variant Master (PVM) and enables better insight and a common understanding of possibilities and limitations in the product family and potential products.

4.2.5 Product configuration

According to Hvam, Haug, and Mortensen [2010], use of configuration/customization techniques and tools to guide users in the task of creating a feasible (and eventually optimal) product specification is increasing. A general definition of the configuration task from Mittal [1989] is:

> **Given:**
> A. a fixed, pre-defined set of components, where a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints
> B. some description of the desired configuration
> C. possibly some criteria for making optimal selections.
>
> **Build:** one or more configurations that satisfy all the requirements, where a configuration is a set of components and a description of the connections between the components in the set, or, detect inconsistencies in the requirements.

A configuration[1] can be viewed as a solution to customer needs by assembling a product from a set of existing pieces/components, be they hardware and/or software components. In later steps after the selection of components, further customization is per-

[1] It should be noted that "aircraft configuration" in the aircraft concept design process means a high-level design of an aircraft (e.g. the type and number of power plants/engines, number of wings/tails, and their positions), but this is not how the term configuration is used in this work.

formed if any of the selected components are configurable. In software, this can be achieved by for example dynamic linking or configuration parameter setting, which is further covered in section 4.2.7 “Binding concepts and binding time” below.

4.2.6 Features and constraints in configuration problems

Feature modeling is used for managing commonality and variability. A *feature* reflects a product capability on an abstract level and is a means for communication between stakeholder and developer in the definition of a product specification, [Sinnema et al. 2006]. The *feature model* represents a set of configurations. The configuration and customization process may be guided through constraint-based facilities, such a constraint checking, propagation, satisfiability, solving, and computing the number of remaining configurations [Chang 2006].

A mature methodology for feature modeling is the FODA (Feature-Oriented Domain Analysis) method. FODA is oriented towards software reuse, focused on domain analysis and its main components are:

- Context analysis: to establish a domain scope
- Domain modeling: to define the problem space, using features, features values, and parameters that provide a description of any real or proposed system
- Architecture modeling: to characterize the solution space.

A description and feasibility study of FODA is found in Kang, Cohen, Hess, Novak and Peterson [1990]. According to Salinesi, Mazo, Djebbi, Diaz and Lora-Michiels [2011], a modeling technique based on constraint programming is more powerful than compared methods, e.g. FODA and OVM (Orthogonal Variability Model). This is due to richer expressiveness of constraint programming; it is possible to use a larger set of domain variable types. FODA is for example limited to Boolean types (“this feature does/does not exist on this system: yes/no”).

4.2.7 Binding concepts and binding time

Binding is a concept that describes that a variation point is bound, i.e. selected to become part of a product variant [Clements & Northrop 2002 and Krueger 2004]. *Binding time* denotes when in the process the variation point is bound. Possible binding times include model time (also referred to as “design time”), code-generation time, compile time, link time, load time, and execution time (also referred to as “run time”). For verified components accessible in a storage or repository, the checkout activity best describes when binding is performed; checkout time[2]. There are similar descriptions for hardware oriented product families, where the term Customer Order Decoupling Point (CODP) is defined as the point where the product is linked to a specific customer order in the manufacturing value chain [Olhager 2003].

Different binding time alternatives may affect the end product system properties, for example security, testability, and performance. A typical creation process related to Software Product Lines with alternative binding times is exemplified in Figure 22.

[2] Checkout time is used here to describe the point where the selection of a variant of the code from several available variants is made, even if the checkout not is a truly "binding" mechanism in software engineering.

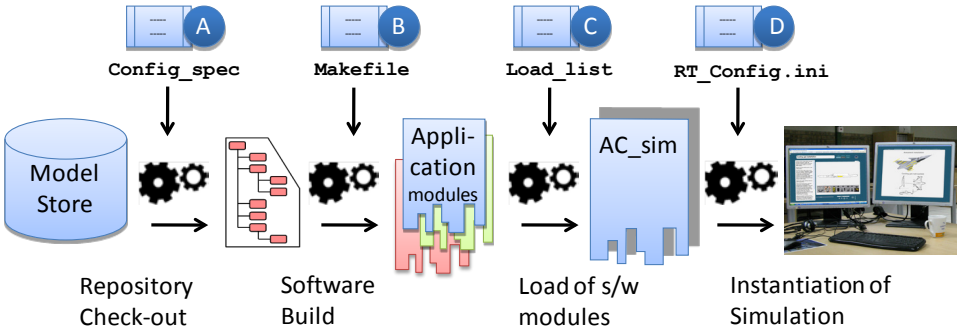

Figure 22. Example of binding time alternatives for configuration of an aircraft simulator instance in a software product line context.

For a simulator product, the simulation model fidelity is an important variation point. A high fidelity model variant is useful in fault simulation and for investigation of the consequences of a malfunction, while a low fidelity model variant can be used to reduce computing performance needs and thereby speed up the execution of the simulation model, thus allowing use of low cost simulators. The actual mix of high and low fidelity models in a simulator configuration depends on what system functionality is to be investigated in the simulation.

In many industrial examples, several binding times are used within the same product family [Rosenmüller 2011]. Late binding (e.g. execution time binding) offers short turnaround time when changing model features. According to van Gurp [2000], there are more reasons why late binding is beneficial:

- The different representations of the system are handled by different people. For example, the architecture design is handled by developers and the running system is handled by users. Some decisions should be made by a user, which means that developers should design the ability to make that choice into the system.
- The needs for new and changed requirements do not generally stop after product delivery. Post-delivery variability techniques help address these requirements more cost-effectively for delivered systems.

There are, however, situations when execution time binding does not meet the information safety requirements, for instance when proprietary models are integrated in training products and delivered to a customer. Only functionality relevant for that customer is allowed to be present in the specific product variant. Thus, customer oriented model variants have to be maintained and binding is performed at checkout time. In addition, for any large mature product family, there will be several legacy components with inherent limitations of binding time alternatives.

4.3 Product families for models and simulators

With a systems engineering approach based on models, it is efficient to reuse the models, and here it is advisable to choose a reuse strategy explicitly, e.g. the copy-and-paste or the product line approach. Bruce et al. report on experience from the synergy result-

ing from combining Model Driven Engineering (MDE) and SPL technologies in the radio domain. MDE refers in this case to the application of Domain Specific Languages.

> *"It is our experience that one big benefit to the software development industry is the combination of the Software Product Lines and Model Driven Engineering technologies."* [Bruce, Paniscotti, Roman & Bhanot 2006]

For a product family with a long lifetime, which is significantly based on simulation results, it is valuable to have mature and well-validated simulation models. These models constitute their own Configuration Items (CI) that also need to be specified and declared for usage, besides the parts/CIs of the simulated products.

4.3.1 Example – simulation model of Saab Gripen's ECS

The Environmental Control System (ECS) in the Saab Gripen fighter aircraft is used here as an example system with related simulation models. It is a complex system that includes both H/W and S/W. Objectives of ECS are to provide sufficient cooling of the avionics equipment, and also tempering and pressurizing the a/c cabin. Essential tasks are also to enable pressurization of the fuel system and to provide conditioned air to the On-Board Oxygen Generating System (OBOGS), which provides breathing air to the pilots. Briefly, this is achieved by using engine bleed air, which is decreased in pressure and temperature and dried prior to distribution. The main H/W components are heat exchangers, compressor, turbine, water separator, pipes, and control valves. The ECS S/W contained in the General systems Electronic Control Unit (GECU), controls and monitors pressure, temperature, and flow levels in various parts of the system. See paper [V] for an ECS layout diagram.

Aligned with the real system layout, the simulation model of the ECS consists of three major model parts, namely the ECS H/W model, the ECS S/W model, and the GECU H/W model. The ECS H/W model is developed in the Modelica language. For more details, see paper [II]. The other two models are developed in Simulink. An integrated model is obtained by using hosted simulation. Both Simulink and Dymola can be used as hosting tools. Characteristics of a planned M&S task determine which tools are most appropriate to use for hosted simulation, which is reported in Steinkellner et al. [2008].

The ECS H/W model has several variants, e.g. one simple and one detailed variant. The model layout is hierarchical and the Modelica construction replaceable is utilized to obtain different variants applicable for model time binding. Additional variant handling is performed by parameter selection at load time and execution time. One view of one of the ECS H/W model variants is shown in paper [V]. In the configuration view, each model variant is treated as a Configuration Item (CI).

4.3.2 Product line mapping

A special property of the set of simulation models representing behaviors and parts of another set of artifacts is its duality, which refer to Figure 7. Provided a simulator configuration and integration process, the set of models constitutes a product line, considered the *secondary product line*. The product set that the simulation models represent, is considered the *primary product line*.

One approach is to map each simulation model to a component and a variant in the product structure. On the other hand, there is the unique flexibility of software modeling, which allows models to be parameterized such that a single model together with a parameter set may be used to simulate multiple component variants. When parameterized models are used the direct coupling between models and the product structure is non-trivial.

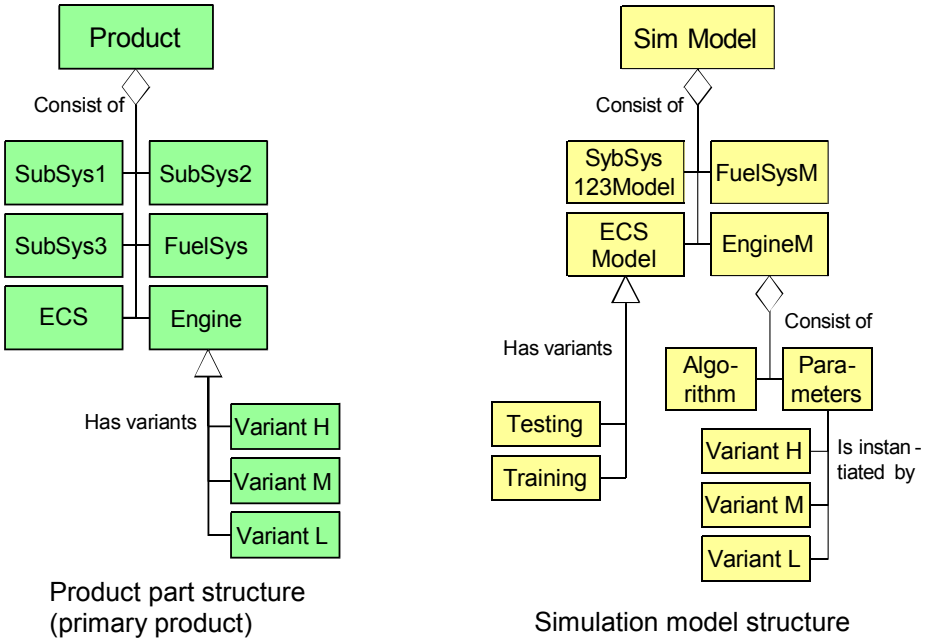

Figure 23. Two different structures illustrating a primary product line and simulation models contained in a secondary product line.

The simulation model structure may not align to the product configuration structure defined in PDM. Two different structure patterns are illustrated in Figure 23. Some reasons for the differences between the two structures are:

- Models have variants for different purposes that do not exist in the PDM part structure, see (Testing and Training variants of the ECS Model) in the figure.
- Separation of the algorithm part from the data part in parametric models, see (Engine ⇔ Engine Model) in the figure.
- One model represents several parts of the product as it is defined in the PDM part structure, see (123Model) in the figure.

An explicit mapping of how and to what degree a secondary product (e.g. an aircraft simulator) represents a primary product (e.g. an aircraft) simplifies maintenance of a configurator system. More details about alignment of structures and the secondary product line in relation to the primary product line can be found in papers [IV] and [VII]. A concept with a structured ConfiguratioN datA object (CNA-string) is proposed

in papers [III] and [IV] as a means to integrate configuration information and to be used for simulation set-up purposes.

4.4 Design and validation of multipurpose models

This section presents a description of a typical workflow for development and maintenance of multipurpose simulation models. Workflow descriptions, used standards, and architecture views should be available to engineers in the form of handbooks and instructions, and in training courses. More details and an example from development of the ECS model at Saab Aeronautics can be found in paper [V]. The aim is to present a way to produce a simulation model of good enough quality to be included in a Model Storage for reuse purpose. The scope of the handbook is simulation models representing physical environment, physical systems, and electronic hardware. Models of embedded software are mainly developed according to other established processes, but some support may be obtained from this handbook. Figure 24 shows an overview of the workflow.

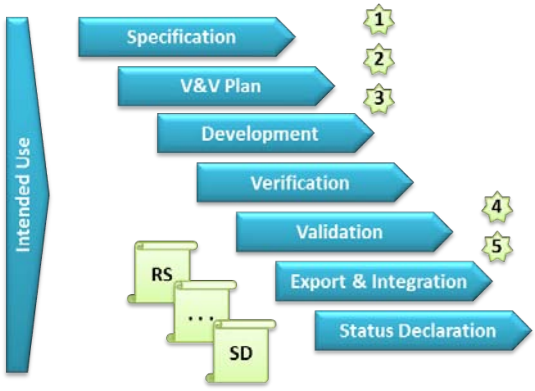

Figure 24. Workflow for development of multipurpose simulation models.

The overview provides a chronological view of the activities in the workflow; however, the duration of each activity may vary significantly depending on the characteristics and intended use of the actual simulation model developed. Another aspect is that activities normally overlap, i.e. the workflow is not sequential.

The stars in Figure 24 represent output items such as code, test cases, or interface descriptions and the symbols named RS and SD represent the documents *Requirement Specification* and *Status Declaration*. The purpose of the SD is to describe the status of the simulation model and its related documentation. It contains among other things information about the model version, its purpose, security classification, and known bugs/limitations.

The purpose of the guidelines in the handbook is also to ensure that code export and integration in applicable simulators in itself does not affect the conclusions of performed tests. The general rule is to place all functionality in the core model, not in integration layers. If there are exceptions, these shall also be documented in the Status Declaration.

4.5 Analyses of product family changes

Analyses are needed of requested and planned changes of a complex product. Simulation is an important tool to validate the changes and to find "side-effects" (unwanted dependencies) that are not identified through other methods. There are some typical situations when the engineering teams use M&S to identify the impact of specified changes on the product family:

- New product variants are defined
- New features are added
- Extended usage of the products.

This applies with respect to changes in both the primary and the secondary product line.

Paper [III] illustrates three different variants of batch mode simulations appropriate for analysis of the primary products:

- when adding a new a/c variant
- when adding new specified usage of the aircrafts
- re-testing existing definitions.

Figure 25 gives an overview of the three different variants. Analyses through batch simulations are performed by executing the model in a specified envelop of operating points by giving initial states and a number of inputs for the dynamic simulation at every operating point. A set of operational scenarios is created with selected values of, for example, speed, altitude, and fuel content as well as a range of pilot maneuvers. It is also a standard procedure to introduce H/W failures (in e.g. sensors and actuators, see section 3.5), in the steady state solution, but also at arbitrary time during the dynamic simulation.

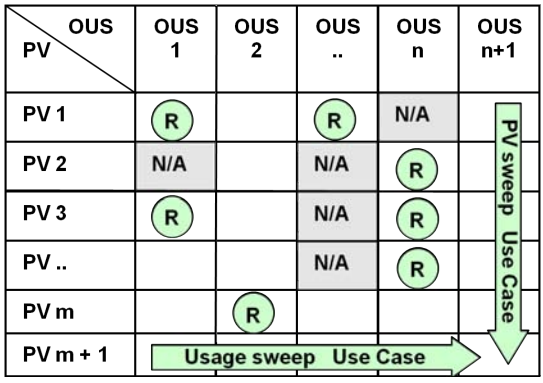

Figure 25. Matrix of Product Variants (PV) versus Operational Usage Scenario (OUS). For an additional PV, analysis is needed of the defined OUSs and vice versa. "R" denotes the defined regression tests to be included in analysis of the full product range.

In batch simulations, the pilot maneuvers (specified by a pilot model) are varied to find the most severe combination of pilot inputs and system failures (the what-if scenarios) in each flight condition and for different payloads. The analysis of simulation results requires a great deal of post processing, for example plotting, comparison between baseline and the current design (two iterations in the process) and summarizing of results.

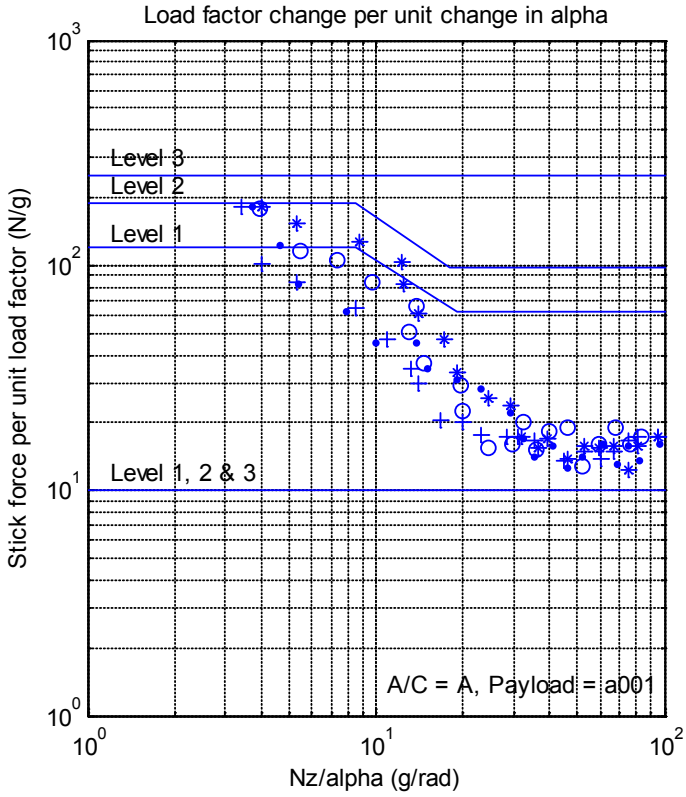

Figure 26. Example of a verification summary plot showing typical analyzes result of a/c Handling Quality properties and their limits (in terms of 'Stick force per unit load factor'). One aircraft type (A) and one payload combination (a001) is summarized in this single plot.

Nomenclature:
alpha: Angle of Attack
Nz: Load factor

Visualization of batches of simulated data can be accomplished by summary plots for an overview and to simplify the analysis work. Results from all operating points (height, speed) for one a/c type and one payload combination are compiled together with information of the limits, for example Handling Quality limits, see Figure 26. For further descriptions of model based development of flight control functions and batch mode simulations, see Andersson and Sundkvist [2006].

4.6 Summary of reuse and its application

An efficient way of creating software or systems is not to develop them, but rather to reuse existing ones. This has been recognized for quite some time and there are different approaches to reuse, for example the lightweight “copy-and-paste” method or the more thorough “product line” method. To instantiate unique products from a product line, there is a need for configuration (or customization) support. In aircraft simulation as in the development of other complex products, not only are well-validated and declared models important, but also verification of the whole simulator product. The simulator is ultimately a central tool to enable safe and reliable aircrafts.

5
Industrial application

THE APPLICATION FIELD of the combination of model based development and product line engineering is aircraft simulators for the Saab 39 Gripen fighter. This chapter describes the product line approach selected, including examples of how simulation models are integrated and used. An overview of existing components and the configuration challenges is given and a prototype implementation of a configurator is described.

5.1 Introduction to industrial application example

In this example from Saab Aeronautics, the simulator product family is partly based on "real" software and avionics parts actually used in real aircraft. This method is denoted the "Design Once Approach" and in Saab Aeronautics' publications, it is stated:

> *"Saab delivers advanced operational support systems and training system media made to reflect the weapon system's current configuration. Saab has established a development process where all requirements for the entire weapon system are captured early, thus influencing its design right from the start. The 'design once' approach, common to all tools and software used in developing the real aircraft, ensures that any changes to the aircraft are automatically reflected in the support and training systems."* [Saab 2011]

The simulator architecture is *modular* and *layered* with simulation models as core components and possibilities to replace model variants by a "plug-and-play" mechanism to simplify replacement and reuse of the models. For Electronic Control Units (ECU) it is possible to use two alternatives: connect the real hardware, or use a model (a representation of the real hardware). The actual simulation code is mostly stored in some source code language accepted by the simulator framework (e.g. FORTRAN, C, or Ada). Models must follow the specified interfaces' guidelines to be easily integrated into the product family, for integration in applicable simulator environments.

There are different ways to introduce or to transit to a product family from an existing "single-products" setting. Many aspects of a transition depend on for example the histo-

ry of and the used methods for product development within an organization. Krueger [2002] describes three approaches for creation of a product line:

- *Proactive:* development "from scratch" with a complete set of common and varying components, feature declarations, and product definitions to support the full scope of products needed on the foreseeable horizon
- *Reactive:* incremental implementation that evolves when new product versions are relevant and/or when new requirements for existing products are created. This approach offers a quicker and less expensive transition
- *Extractive:* the organization uses existing custom products to extract the common and varying components into a production line. Assets are identified during the extraction. This approach for reuse enables the fastest adoption.

In the simulator application example, the *reactive* approach was earlier used for each simulator family (development-, verification-, and training simulators). For the common product line effort with a larger scope, where merging of model code from existing simulator repositories is a major concern, the *extractive* approach best describes the situation. For further description and definition of the application problem, see studies of Saab Aeronautics simulation model product line efforts in papers [III], [IV], and [VI].

5.2 Simulation models

Most of the aircraft functional systems and parts have representations in the model domain. A model inventory was performed and an overview was developed to gain a common understanding of the models included in all simulator types. The following classes of models were defined, with reference to the overview in Figure 27.

- Models of aircraft surrounding, classified as "Physical" in the figure, are normally not included in the product part structure. Example: the atmosphere model,
- Models of aircraft parts (mechanical assemblies, sensors, or whole aircraft subsystems) classified as "Aircraft". Example: the engine model,
- Models of avionics product parts (e.g. electronic control units, ECUs) classified as "Avionics". Example: the Air Data Computer, ADC,
- Models of on-board embedded software classified as "ACSSW" (Aircraft Computer System SoftWare). Example: the navigation software,
- Models of payload and stores that may not be included in the aircraft part structure but are separate products classified as "Stores". Example: the drop tanks.

The "Avionics Core" block in Figure 27 contains both avionics hardware and software kinds of models. There are also some special models, for example pilot behavior and tactical scenarios.

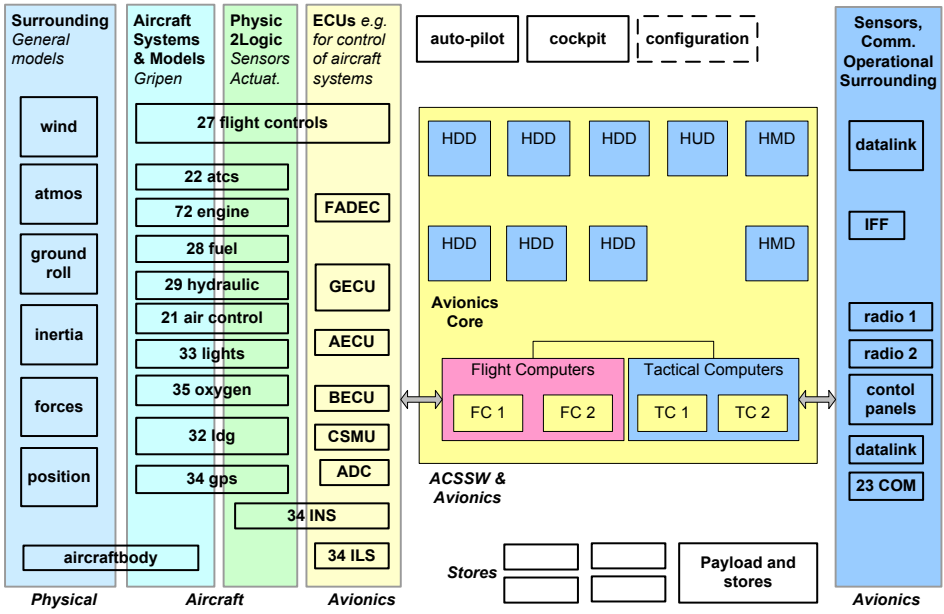

Figure 27. Classification of reusable multipurpose simulation models. Abbreviations are aerospace specific terms, e.g. Electronic Control Unit (ECU), Head Down Display (HDD), Helmet Mounted Display (HMD), Air Data Computer (ADC), and Identification Friend or Foe (IFF).

Because some models are used for several aerospace product families, the assets (models) reach across product families. Examples are the atmosphere and wind models, which are used for simulation of all airborne products (e.g. UAVs, fighter aircraft, and space vehicles) at an enterprise level. The study, however, is limited to the 39 Gripen product family, and all models are assumed to be fully managed within that scope.

5.3 Legacy and third party components

There are specific challenges with the introduction of model based methods and tools for existing products and in the existing development environment. The new tools provide a range of model types, notations, and languages [Andersson 2009] and these "new" models together with "old" or "legacy code" models have to be integrated. For models in the form of legacy code, design information and test cases are documented in a variety of different formats. Some are not electronically available, which complicates documentation updates. Transformation of all existing legacy models into new formats is at present not considered to add sufficient value.

Some of the simulation models are developed by partners and suppliers of equipment or a/c subsystems, which is the case with for example the high-fidelity variant of the engine model. Saab does not have full insight as regards how these models are implemented, but relies on the sub-supplier organization through their model validation activities and supporting documentation. Another similar situation concerns some "standard" models, for example the atmosphere model, where the specification is adopted from ISA

(International Standard Atmosphere) [ISO 2533:1975]. Yet another kind of component, including tools for software unit test, could be classified as third-party software components.

The size of the reference configuration of the a/c Simulation Models part, including legacy, third party and newly developed components, is of the magnitude 1 MLoC (Mega Lines of Code)[3]. The reference configuration is a typical large-scale build and contains code for simulation models + adaption + connection layers. An additional up to ~2 MLoC embedded software (in a/c core avionics) is included depending on a/c version and variant. The connection and adaption layers (see paper [V]) are implemented in the Ada language. Older "legacy" code is mostly developed in FORTRAN and C. Generated code from modeling tools is generally in C/C++, and the generated code is often more numerous compared with hand-written code. See Figure 28 for the relationship between different implementation languages.

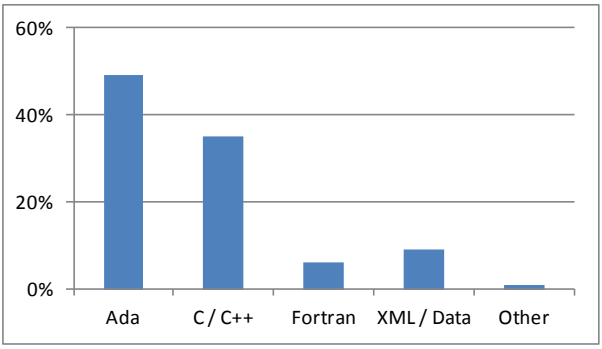

Figure 28. Relationship between languages used in the implementation of simulator models and additional components for the reference configuration of an aircraft simulation kernel.

Parameter files and other datasets are traditionally stored in simpler text files with parameter values separated by for example spaces. Newer formats for data are based on XML, which enlarges the code "volume" due to its mark-up principles.

5.4 Configuration and customization needs

Several different user needs drive the need for configurator support as described in the appended papers [III], [IV], and [VI]. One example of simulator customization is the fidelity level of a single simulation model. This may be implemented as strictly separate model variants in the Model Storage, which may increase maintenance costs for example during model enhancements and error corrections. One way to avoid several different model variants is to use the Multi Level Approach reported by Kuhn [2008]. With this approach, switching between fidelity levels can be done within "'the same" model, so one single model variant may serve different purposes, from example:

[3] 'Lines of Code' is a software measure that is suitable when comparing the size of different source code bases. It should not be used for absolute estimations of e.g. needed effort for test. [Park et al. 1992].

- a simple and fast model for energy consumption design
- a detailed model for fast network stability analysis
- a detailed model for network quality assessment by increasing the equation complexity in the model components

There are disadvantages in using the Multi Level Approach: for instance, the interface definitions need to be the superset of all required signals for all levels. If several interchangeable model variants are used in a "plug-and-play" approach, the interfaces also need to be the same at all levels of detail (viewed as a black box). One of the most important preconditions of the configurator system is the existing instantiation and integration process. An overview of the process is found in Figure 29. The main parts include:

- A repository for storage of validated simulation models and other components needed to build an aircraft simulation,
- A build system with a predefined directory structure and software build tools.
- A transfer mechanism including packing of executables, moving the packages to the simulator environment, unpacking and simulator installation scripts.
- Initiation of the simulation models in the target simulator. Many of the models have configuration parameter inputs that are set from parameter files and/or by the user, depending on the kind of simulator.

The overview also shows the flow of user requirements and constraints through the configurator tool and further to specifications as input for each binding mechanism. To get an understanding of the level of flexibility in the system, let us calculate the number of combinations based on numbers of alternatives in the simulator product line.

Without taking into account any constraints (in terms of configuration validity), the multiplication principle is used to calculate the total number of possible combinations for selection of exchangeable components. If there are a ways to select A and b ways to select B, then there are a times b ways to select A and B. For n number of selections the total number c is

$$c = \prod_{i=1}^{n} z_i \tag{1}$$

The variable z_i in equation (1) is the number of existing choices (versions and/or variants) per component class i.

At one point during development, there were 76 classes of components, some of which are available in variants and/or versions. There were for example nineteen classes with two available alternatives and one class with six available alternatives according to (2).

$$c = \prod_{i=1}^{76} z_i = 1^{46} \text{ x } 2^{19} \text{ x } 3^{8} \text{ x } 4^{2} \text{ x } 6 = 3.3\text{x}10^{11} \tag{2}$$

The total number of combinations at that time were: $3.3\text{x}10^{11}$. The high number is referred to as the principle of *combinatorial explosion*.

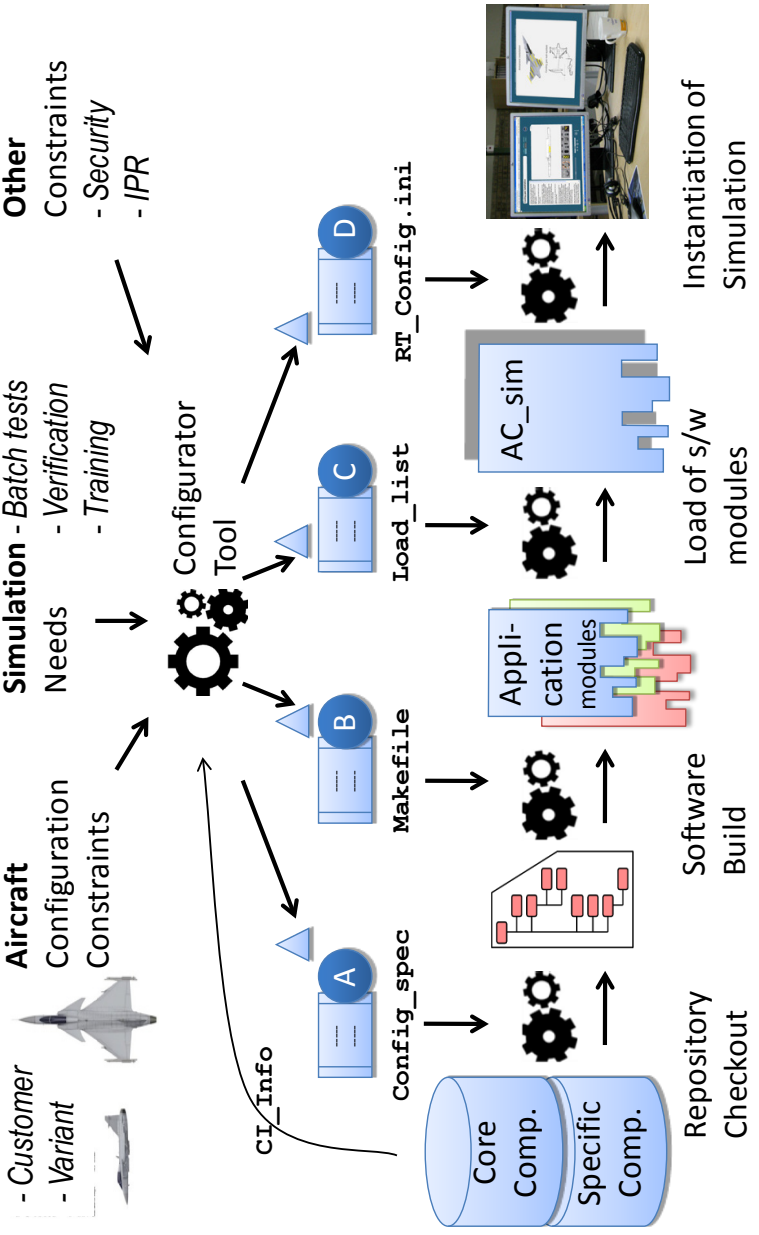

Figure 29. Means for control of configurations in the simulator build process. Aircraft configuration constraints are interweaved with simulation needs.

Nomenclature:

IPR	Intellectual Property Rights
AC_sim	Aircraft simulation
RT_Config	RunTime/execution time Configuration parameters

5.5 Configurator prototype

A prototype configurator implementation is created in order to obtain a proof of concept of the configurator principles, including interfaces to surrounding information systems. The primary objective of configuration support is to increase the degree of reuse of models and test assets in order to reduce lead-time for creating new simulator configurations, and to increase the quality of specifications of simulator configuration. Development of the configurator prototype was performed in increments. The concept phase utilized a phenomenon model in the form of a Product Variant Master (PVM); see the appended paper [III]. An information model formed the foundation for a prototype at the university; see paper [IV]. The configurator prototype in the industry site is described in papers [VI] and [VII]. Here follows a summary and some additional comments.

5.5.1 Basic usage scenario

A set of usage scenarios was created in order to obtain a common understanding of how to prioritize functionality of the configurator system and to create an incremental implementation plan. The basic scenario covers a request from an aircraft development team who needed a minor update of an existing simulator configuration in two respects;

1. a model modification due to an aircraft change (based on a primary product decision)
2. a replacement of a model from a high fidelity variant to a low fidelity variant (driven from a secondary product decision)

As the Gripen aircraft development at Saab Aeronautics is in practice incremental, there was a fundamental requirement; it shall be possible to start from an earlier configuration and from there make minor changes to get a new valid configuration.

5.5.2 Metadata structure and storage

Data about components are managed through a data storage object called **`CI_Info`** (short for Configuration Item Information). For example, the data for all variants and versions of the ECS models are collected in one file. The data in **`CI_Info`** is separated into two categories;

- structured data which is possible to use in the configurator inference engine. Boolean, integer, float, and enumerations were used for this
- data in text format used for searching and to generate documentation, for example component lists and a Release Notes document for each component release.

`CI_Info` objects are stored in XML-based files, one per component class. A component class is here defined as all variants and versions of a component. All the **`CI_Info`** XML-files must comply with the specified structure, and they are validated against an XML-schema during execution of nightly build. The data is stored in the same source code repository as the simulation models and is fed into the configurator model as shown in Figure 29.

The **`CI_Info`** data is a central part of the configurator model and its implementation. The files are *included* in a configurator *main model* through an include function. This enables management of each **`CI_Info`** as an object in its own right (in fact a Configura-

tion Item of its own) with its own lifecycle, and updates to the configuration model can be made concurrently by several developers. The constraints specifications (rules) and view settings for the user interface are stored in the main model. Components (variants and versions) and features can be used to create rules, for example on how the models and connectors can/cannot be combined.

From the first iteration, great attention was paid to maintainability of components, attributes, and constraints in the feature model. A distributed approach was chosen with the involvement of the developers for updating of information in the feature model when a new model version is due for release. The information model was created and decided in the development team so that stringent naming principles for example were implemented to simplify maintenance.

Text attributes in `CI_Info` enabled automatic generation of Release Notes documents and Configuration Item lists (variants and versions of components/models). In the previous routine, separate documents and spreadsheets were updated in a document management system by manually "filling in" of data, which is tedious and error prone.

5.5.3 Build process and binding time

Simulation needs and aircraft configuration constraints meet in the configurator where knowledge about combined constrains is stored; so that possible and incompatible model configurations can be visualized for the user, see Figure 29. When a desired configuration is found and accepted by the user, the result is transformed to four different specifications used for configuration control during the build and instantiation process. The specifications are used at the respective binding time in the process:

A. Checkout list for variants and versions of source-code from the repository
B. Make file for the software build (compiler and linker) used at compile time
C. Load list with specification of variants for the loadable simulation components
D. Parameter file for initialization of simulation model through `init`-parameters

To verify that a set of *reference configurations* are valid, a nightly build procedure is implemented. During the night, scripted tools complete some or all of the steps in Figure 29 depending on availability to do automatic load and simulation in the respective simulator. As result of the nightly activities, a status report web page is created where a summary of each build/run is published. The Nightly Build Report contains, for example, the following information:

Simulator Variant	**Build Statistics**	**Unit test Statistics**	**System test Statistics**	**Log Status**
Config 1	pass	pass	fail	ok
Config 2	pass	pass	pass	ok
:				
Config n	pass	fail	n/a	ok

5.5.4 Tool support

The “Tacton Configurator Studio” tool developed by Tacton Systems [2011] is used as configurator and inference engine. The decision to use a commercial and mature tool

instead of developing a new one is based on the focus on integration of information and systems rather than on tool development.

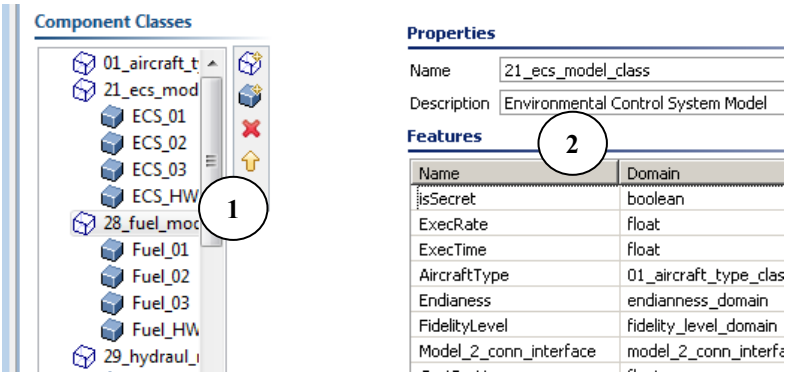

Figure 30. Parts of the configurator implementation; 1) Component database and 2) Feature specification

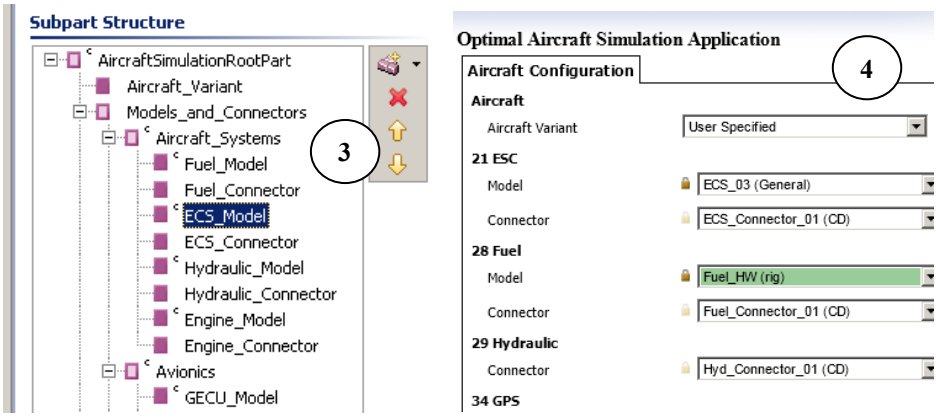

Figure 31. Parts of the configurator implementation; 3) Part structure model and 4) Configurator user interface.

The different parts of the configurator tool are described below with reference to the screenshots in Figure 30 and Figure 31.

1. **Component database.** The database comprises classes of components such as aircraft, simulator, simulation model, connector, and simulator interface.
2. **Feature specification.** Variation points are modeled as features. Features for the simulation model classes include:

Name	Description	Unit/Value
IsSecret	Security classified	`[yes/no]`
ExecRate	Execution rate	`[Hz]`
ExecTime	Execution time per iteration	`[s]`
AircraftType	Product Variant usability	`[aircraft type]`
Endianness	Software endianness	`[little/big]`
FidelityLevel	Level of model fidelity	`[1..6]`
ReleasLevel	Release Level / Maturity	`[alfa/beta/full]`

Mandatory features are security classification, usability, and maturity.

3. **Part-structure model.** The principle of the configurator part-structure is to reflect the grouping of simulation models found in Figure 27. Aircrafts and simulators are defined by only one part each. This view of the configurator implementation includes the set of rules specifying the constraints on how the models and connectors can/cannot be combined. There are also constraints defined in the form of calculations, for example for predicting the total execution time for all models included in a configuration.
4. **Configurator user interface.** The Tacton Configurator Studio built-in user interface is used in the prototype. Selected variants are "locked" with a padlock symbol. Possible choices are marked green and prohibited choices orange. For every selection, the inference engine recalculates further possible choices based on current selections.

Component types and features may both be used when creating rules and constraints.

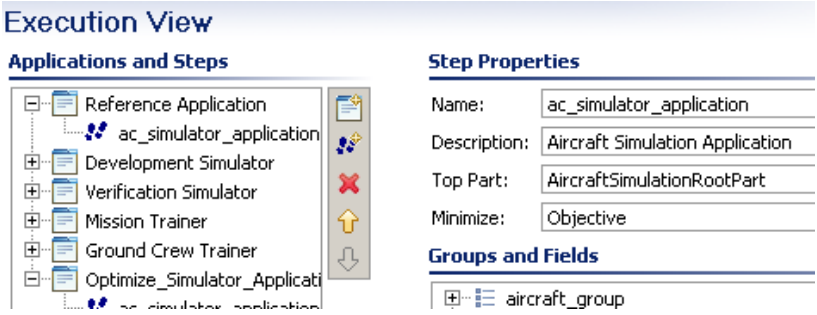

Figure 32. Several predefined applications are used with a customized set of selections for each application. An application corresponds to a type of simulator, e.g. a simulator intended for verification.

The simulator type definition (development, verification, and training) is a classification of simulators, so each class can be defined as a family aimed for an application field. It is convenient to use pre-defined application definitions with a customized set of choices for each application, see Figure 32. Only detailed feature selections relevant for a simulator type (application) are visible in its configurator user interface (screenshot 4 in Figure 31). The non-visible features are either pre-set for that application or calculated by the inference engine.

The Tacton Configurator tool supported by XML/XSLT tools and the Python [2010] scripting language are used for the prototype. For a full-scale implementation, a SWOT (Strengths-Weaknesses-Opportunities-Threats) analysis is initiated to support the tool selection. This analysis includes Tacton and two alternative tools oriented towards the Software Product Line market segment: "Gears" from BigLever [2011] and "pure::variants" from pure-systems [2011], [Beuche 2008]. These two are promoted as having integrations to systems and tools for requirements management, software development, and Software Configuration Management, but they are not considered being as mature as the Tacton tool.

The EU project ConIPF (Configuration in Industrial Product Families) has developed and made a good description of a tool independent methodology that is aimed at "*support product derivation during application engineering with a combination of product line engineering and knowledge-based configuration*" for software intensive systems Sinnema et al. [2006]. ConIPF and also Munir and Shahid [2010] have compared tools for SPL and the tools that both reports have in common are "Gears", "pure::variants", and "MetaEdit+" from MetaCase [2011]. Further evaluations of methods and tools, must be performed in the next step before a full-scale implementation can be made. The prototype implementation has provided good experiences and shown that it is possible to build and use a configurator system for a mixed system/software-based product family with a non-software oriented tool.

5.6 Summary of the industrial application example

Transformation from product centric development/support towards a product line focus has been going on for several years with the 'design once' approach as a vision. The modular simulator architecture provides a basis for flexibility and enables models to be replaced in a "plug-and-play" fashion. Similar model variants from existing simulator platforms are merged into common reusable multipurpose models. The large portion of legacy components (mostly validated simulation models) imposes constraints on the possibilities to define configurations. Development of new models with modern tools/techniques enables variability through, for example, inheritance in object oriented languages. As the reusable models and other artifacts grow in number and more simulator variants can (re)use models from the common Model Storage, the need for support for creation of stringent customized configurations increases.

To provide a solution proposal for this need, a configurator tool to customize and support integration of simulation models for different types of aircraft simulators has been specified. A concept for managing configurator data is selected based on needs, existing methods/tool chains, as well as information from performed interviews & inventories of models. Classification of models, definitions of use cases and features are established within the team. Component-data, features, and constraints are implemented in the configurator prototype, in XML-files, where a set of constraints in the primary product family (aircraft) is used for the configuration of the secondary product family (simulators). Beside the features, other information about components is managed in the same XML-files for generation of for example Release Notes documents for new component versions.

6 Results

RESULTS FROM THIS research include experience from industrial usage of emerging modeling languages and an initial implementation of a configurator system in an industrial product family for aircraft simulation systems and products. To obtain qualitative data for evaluation of the solution, information was collected based on the participants' experiences according to the iterative, retrospective method Scrum as described in Chapter 2. The experiences from using new languages are reported in papers [I] and [II]. The problem domain and initial solutions for reuse of simulation models are covered in papers [III], [IV], [V], and [VI], and results from collection and analysis of the validation data are summarized in paper [VII]. Here follows a summary of the results and additional comments.

6.1 Industrial experiences from modeling languages

Experiences are reported here from support of modeling techniques based on three emerging languages/tools: Modelica, Simulink, and SysML. These are interesting for industrial, large-scale development, and are related to the application of simulator products covered in this dissertation.

6.1.1 Modelica and Dymola

For simulation of vehicle systems, the introduction of the Modelica language and the Dymola tool [Dassault Systemes 2011] at Saab Aeronautics has largely been positive. However, there are several areas where method and tool support must be improved before Modelica supported by Dymola will be natural to apply in projects developing complex vehicle systems. Areas that need improvements, which are also reported in paper [II], include:

- support for large-size models
- support for model uncertainty and quality tracking
- support for validation of complex models using measurement data

- features for set-up, execution and post-processing of batch simulations
- structure of generated code from Dymola
- performance of generated code (important for real-time simulations)
- code generation support for multi-thread and multi-processor targets
- generation of model documentation adapted to industry/aerospace standards

The evolving integration of mechanical design and Modelica M&S through tool providers consolidation described in section 3.4.2 has potential for improved collaboration and efficiency. This would enhance automated dataflow from the specification to analysis models (covered in section 3.3.2) and further support the model based approach. Dymola version 7.3 was used for the results reported in paper [II].

6.1.2 Signal flow modeling with Simulink

Capabilities and limitations of the Simulink toolset have been evaluated, in the planning and concept study phases of new variants of the 39 Gripen aircraft, to explore how the modeling tool/technique can support model based systems/software engineering. In Andersson, Weitman, and Ölvander [2008], three different approaches of Simulink usage for functional development are presented:

1. A functional oriented modeling approach where simulations of the functions are in focus.
2. An implementation oriented specification approach based on a modeling framework with predefined system architecture, scheduling, data types, and rules for discretization. With this approach, the final embedded software is hand-coded using the models as a specification.
3. Similar to approach two, but the embedded software is automatically generated using a code generator suited for production of embedded software.

The reported experiences are focused on prerequisites concerning scalability, such as; model architecture, license model, and project ramp-up challenges. The results are also compared to the existing SystemBuild based development environment reported in Andersson and Sundkvist [2006]. When introducing high-end engineering practices and tools such as Simulink in an organization developing safety-critical products, it is important to make sure that basic management practices (e.g. Requirements, Configuration, and Change Management) are thoroughly handled.

The conclusion is that no one of the studied approaches is superior. In the existing environment there is a tradeoff between state-of-the-art methods/tools and traditional methods for legacy components. Approach number two is most appropriate for quick ramp-up because it is similar to the existing method, which is based on the SystemBuild tool. It is also more flexible, compared to approach one, due the absence of an advanced code generator.

6.1.3 Systems modeling with the SysML language

In the area of systems modeling, the modeling notation SysML has been introduced together with the tool Rhapsody [IBM Rational Rhapsody 2011]. The methodology was evaluated in two development projects at Saab Aeronautics:

- New development of the unmanned Skeldar rotorcraft, see paper [I].
- In development of a new variant of the 39 Gripen system which is reported in Herzog, Andersson and Hallonquist [2010]

The major findings and recommendations can be summarized as follows:

- Introduction of SysML has largely been positive. The main contributing fact for the success is undoubtedly the training program instigated for the project teams.
- The potential of SysML is the greatest obstacle; it needs to be substantially limited/tailored for large-scale usage. It is important to develop adequate modeling guidelines that clearly describe what information should be captured in SysML and what should be captured using traditional methods. In the absence of such guidelines, users have a tendency to add information to the model just because the possibility exists, leading to information inconsistency and redundancy.
- A modeling method/tool needs to co-exist with non-model based methods, tool, and existing infrastructure. A recommendation is to ensure that co-existence is as simple as possible for interfacing tools and for the groups of engineers affected by the modeling/models but not directly involved in the model based parts of the development work.
- SysML is weak in capabilities needed for variant management, product configuration, and for evolving and maintaining a set of realized products and systems. With its roots in software engineering, it is natural that system specification in SysML facilitates the interface to tools for software development (as this is typically performed in the same tool and stored with the same format as system design). Interfaces to tools used in other engineering specialties are, however, weak.

The SysML implementation in IBM Rhapsody was not yet stable at the time for evaluation. During updates to new versions of Rhapsody, engineers needed to go through and possibly update all models in order to maintain model consistency in the new tool version. The document generation software delivered with IBM Rhapsody was very powerful, but was not always consistent in finding model elements. There are several areas where method and tool support must be improved, especially reduction of language and tool complexity, before modeling with SysML/Rhapsody will be the natural method to apply in projects that develop complex systems.

6.2 Reuse and customization of simulator products

This section summarizes results gained from on the prototype implementation of the configurator system to support customization and instantiation of simulator products. Results include model inventories, analysis results, the underlying meta-models, usage guidelines, and evaluations from initial usage of the prototype. Challenges in the described industrial application due for example to many variants and versions of simulation models are reported in papers [III], [IV], and [VI]. Results of the validation data are reported in paper [VII].

6.2.1 Insight and understanding of product line challenges

The explicit and visual description of product variants, core assets, and their variability through the Product Variant Master (PVM) and the configurator prototype enables in-

sight and better understanding of the complexity of single products, but more importantly, of the union of products. Interest in configuration and customization issues has increased in the simulator team because of the configurator prototype (and demonstrations thereof). For the same reason, the team responsible for product data management has increased their coordination activities towards the simulator domain.

Scoping of the product line in terms of a defined simulator family, the set of simulation models to re-use, degree of commonality, and how to limit the domain ranges, has been highlighted. Two examples of limitations of the scope are:

- Old and existing (delivered) a/c configurations are excluded from the scope
- Simulation configurations for concept evaluation, or "early validation" (which was in the scope of paper [VI]), are not covered at present.

Reduction of the original scope gives a clearer feature model, and limits the maintenance. There are still sufficient challenges and enough value in the limited scope.

6.2.2 Binding time differences

The binding time concept was shown to be more important for the configuration and customization activities than expected when the problem was first formulated. A difference in binding-time needs between in-house simulators (for development and verification) and delivered simulators (for training), was visualized through the work with the configurator system. This refers to section 4.2.7 “Binding concepts and binding time” and the question; who is responsible for selecting features; the developer or the end-user? Late binding provides more flexibility for the simulator end-user, as illustrated in Figure 33.

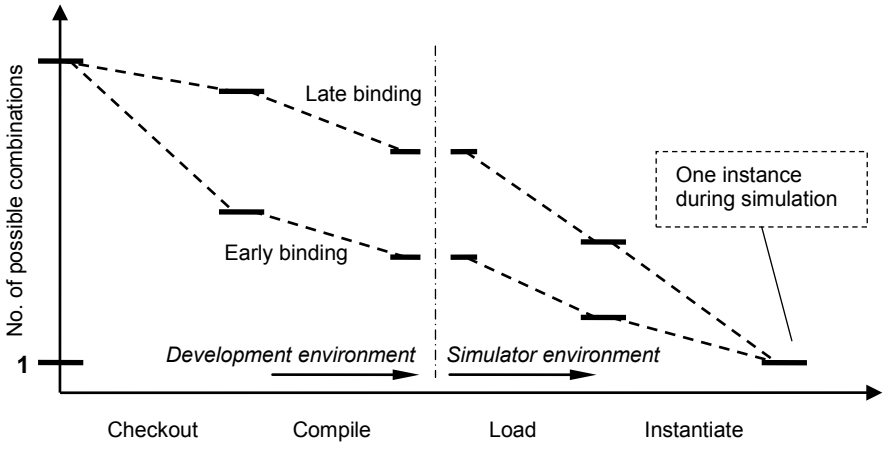

Figure 33. Each step in the build-and-instantiation process reduces the number of remaining possible combinations. Binding (early) in the development environment restricts the possibilities to configure or select features in the simulator. Late binding provides greater flexibility for the simulator users.

There was an increased insight among system architects and model developers of the principles of binding time, and the characteristics of end products depending on the binding time alternatives for different features.

In the final step of the simulator instantiation process, the terminology is confusing. The term "run time" in software engineering denotes a situation where the software application is running or executing. In the simulation domain, "run time" means that the simulation is running and calculates new states and outputs at subsequent simulated points in time. When the simulator is halted, "time is stopped", but the software application is still running/ executing and from a software perspective, it is still in "run-time". In this dissertation, "execution time" refers to a software perspective, so the simulator could be either started, stopped, halted, or any possible mode as long as the software application is "alive". "Run time" refers only to simulation with simulated time evolving.

During specification and design of new multipurpose simulation models, there was an increased focus on what binding time solutions to use. The configurator tool and its associated information model were used as catalysts for the architectural design of binding time for features, simulation models, and for the software-build system. This contributes to better-specified multipurpose models in respect of instantiation, and is thereby an enabler for efficient reuse.

6.2.3 Practitioners' experiences

Guidelines were created to provide support to team members in the updating of components, features, constraints, and rules in the feature model. Initial experience of incorrect data resulted in the development of tools/scripts for automated data validation against available data in the software repositories.

Tool functionality enabling *save*, *restore*, and *make modification* to a configuration (during the development of a configuration specification) were stressed by users with configuration management responsibility. This is because incremental development and smaller changes resulting in many iterations and baselines is a standard procedure in aircraft development (e.g. at Saab). This functionality was not yet designed/implemented in the prototype.

There was a concern about tool performance of the configuration task during selection of features and components. In the relatively small prototype models, calculation time was negligible, and for the mid-size models tested so far, no significant problem is observed as regards performance as long as the recommended data types are used in the feature models. The recommendation is to avoid using the `float` type widely if the model is large, which is due to the combinatorial problem of continuum in the inference engine.

It was found that the software developers prefer file-based software revision systems, while roles with CM (Configuration Management) responsibility find CM systems based only on such techniques limited. File-based systems lack rigor functionality regarding management of for example changes, baselines, releases, permissions, and other metadata. One example of a limitation in a file-based system is that names are used as unique identifiers and it is cumbersome to change names in a file-based system compared to an object oriented CM system. With the existing solution (commercial tools mixed with scripts and smaller tools), there is an identified risk of unsupported integra-

tion and/or high in-house maintenance cost. This has also been observed by Crnkovic et al., who compare SCM with PDM systems: "*In most SCM systems, however, there are only triggers, which can execute scripts written by the users. This is support of little value, as the result is a mess of scripts that are difficult to survey and maintain*." [Crnkovic, Asklund, and Dahlqvist 2003].

All data for the configurator system in the industrial application is under version control, but to ensure quality and consistency of the features and constraints, a change management procedure should be implemented.

The introduction of the `CI_Info` objects for storage of model information, provides a basis automatic generation of the Release Notes (RN) documents and Configuration Item lists (CI-list), and is an clear improvement. The previous routine, based on a separate document management system, included manual "filling in" of data in documents, which is tedious and error prone. Practitioners' experience is positive because "copy-and-paste" is replaced with product line automation. In the questionnaires, the respondents answered that by using the configuration system, the effort/time was reduced. Three respondents had experience using both the previous and the new system. They estimate on average that 30% of the time is reduced to create the RN documents. To create CI-lists, 60% of the time is reduced, provided the information has been added for RNs. All respondents believe that the increased quality of artifacts is a more important improvement compared to timesaving and simplification-of-work.

6.2.4 Domain integration

Activities with inventories, model classification, feature elicitation, and population of data in the configurator played a central role for the connection between the primary (aircraft) and the secondary (simulators) product lines. Two means provided a direct basis for discussion and collaboration among aircraft subsystem responsible and product line engineers who are maintaining and integrating simulation models:

1. Visualization of compatibility between components, aircraft variants, and simulator applications via the configurator user interface.
2. Publication on the intranet of tables of component data generated from the `CI_Info` objects, which is SCM/PDM rather than configurator functionality.

Effects of aircraft configuration constraints on the simulators are made explicit so that the creation of simulator configurations with respect to a/c variants is apparent in a new way. The low level of alignment between product structures, as described in section 4.3.2 became more visible. Of the different simulation model types, the *aircraft computer-software* model and *avionics* model types are well aligned.

The concept of a structured ConfiguratioN datA object (CNA-string) proposed in papers [III] and [IV], to be used for simulation set-up purposes, was reviewed. The result showed that a similar mechanism already exists for software configurations (aircraft editions). For data in the PDM domain, a large effort is needed to create the automated mapping, because there are different identities and naming conventions in the two domains. The amount of configuration data in PDM is much larger than needed for simulator set-up, so still a manual work of mapping is made.

In the modular system with all possible combinations (3.3×10^{11} in the example from section 5.4), the configurator prototype is a catalyst for creating explicit rules for compatibility between components. The rules (based on features and expressed as constraints), is in the first step documented in a spreadsheet before being implemented in the configurator. This procedure was introduced because the configurator is still a prototype implementation, and which configurator tool to use was not finally decided. Explicit information about compatibility is valuable regardless of configurator tool.

6.2.5 Reuse potential and multiple product lines

There are different development teams for simulator applications at Saab Aeronautics (for development, verification, and training simulators). Each of them manages in practice a family of simulators. Typical choices available in most simulators are:

- customer variant
- setting of single/dual seater aircraft
- selection of aircraft software edition

The product line described in this dissertation, consisting of three simulator families, is in fact a product line of product lines ('multi-product line'). Taking the a/c development in the scope, yet another product line (a part of the primary product line) can be identified; the avionics embedded software components, which are used both in the a/c and in the simulators. Some of the larger simulation models may also be viewed as product lines of their own. For example, the Environmental Control System (ECS) model exists in four variants, which all have configuration input parameters for different settings. The low fidelity (LoFi) variants are used for simulations where the ECS system not is in focus, but the model is still needed to be part of the system. The high fidelity (HiFi) variants provide more details and better accuracy, but require more execution time from the simulator time-scheduler. By using the same customization principles and same configurator tool for several product families the reuse can increase between product families, thus throughout the multi-product line.

7 Discussion and Conclusions

MANY EVALUATIONS of methods within model based development use comparatively small applications examples. These are usually applied to real engineering problems, but are demonstrated by small or well-suited examples (compared to the industrial cases) and may not be included in a context of several methods or tools. In industry, terms like *model set*, *tool set*, *tool chain*, and *methods chain* are nowadays used to define aspects of state-of-the-art engineering environments. This work is based on the "industry-as-laboratory" research approach and has a broad scope regarding implementation and evaluation in an industrial environment. However, it does not go in any depth into the technical details of, for example, protocols, formal methods, or computer architectures.

7.1 Discussion

Management of multiple product families needs enhanced support compared to single product line engineering, as described in Section 6.2.5 "Reuse potential". Support is needed in the following areas:

- A recommended product breakdown structure
- An appropriate responsibility structure
- Architecture of the feature model(s). Should it be central or distributed?

A mapping of the primary and secondary products to the ANSI/EIA-632 [1999] system structure (see section 3.2.1), provides a view of how a single simulation model can be used in several system products, see Figure 34. The system structure shows how the simulation models for development (to the left), verification (to the right), and training (in the middle) are dispersed. It is not only simulation need that may differ. The requirements concerning test, documentation, and declaration of models are also different

depending on context and product. This stresses the need for a merged set of requirements and principles for the model development process if the model implementation is to be reusable efficiently.

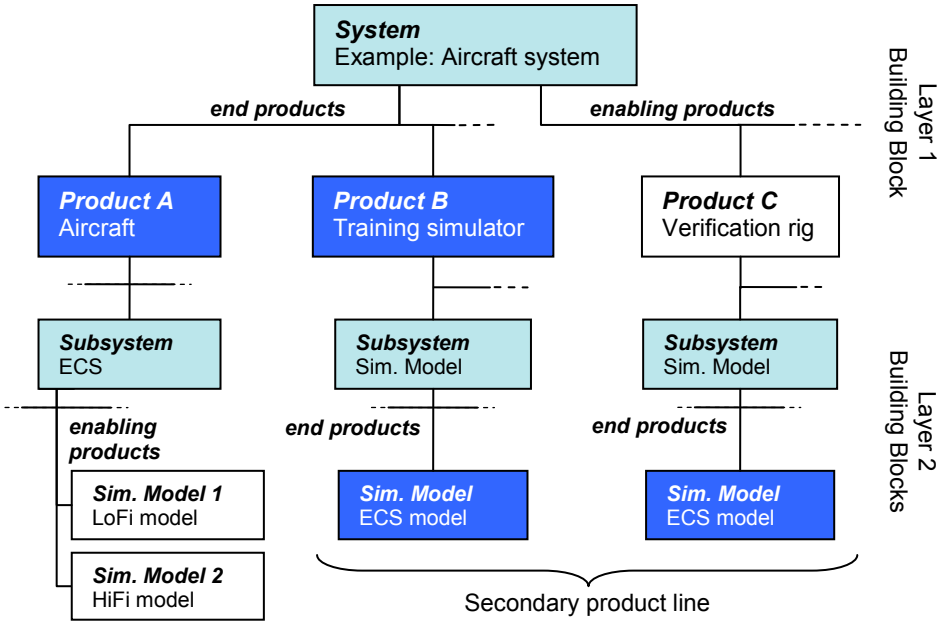

Figure 34. A structure of systems and products according to the ANSI/EIA-632 standard. Examples from the industrial application, where ESC is the Environmental Control System, and LoFi and HiFi are respectively low and high fidelity model variants.

One issue under discussion is the introduction of model functionality. Is it optimal to create a multipurpose model including all requirements and functions from the beginning, or is incremental model development starting from a single-purpose model more efficient? An incremental method enables validation of basic functionalities in the first release, and extension of the model in later iterations. Simulation models for product development should be available early. Elicitation of "all" requirement requires effort and time and delays the model. The delay, which a full specification may cause, will thereby prevent an efficient model-based methodology. In lean development, Just in Time (JiT) deliveries are used, for example implementing training functions in later iterations. An incremental process that supports this should be formed, where each model delivery has an explicit purpose, for example, development, verification, and training. However, the architecture must be properly defined and the basic requirements should be known from the outset. The standard *ARINC-610* [2009] "*Guidance for Design of Aircraft Equipment and Software for Use in Training Devices*" is intended for training simulators and thus relevant for the specification of multipurpose models. Using a standard like this is not a guarantee, but provides good assistance to succeed to specify and build multipurpose simulation models.

The commercial tool Tacton Configurator was selected for the configurator prototype. Tools like Tacton have a majority of users in the domains of design automation, product

customization, and sales configuration with integration to CAD and hardware design. Central model properties in this type of tool are component price, choice of currency, and part number attributes, and output from the configuration activity is typically the Bill of Material (BOM). Those attributes and outputs are not of central interest as software properties, and the tool type is therefore not adapted to support a pure software product line. There is a risk that a configurator system supported by this kind of tool does not scale up for customization of software products. In the studied application, there is need for support for both Product Line / Software Product Line (SPL) engineering so the choice of configurator type is still regarded as adequate. In practice, two types of tools for product line support are identified; tools like Tacton and those who support SPL. A tool type that supports both would be needed.

A mandatory discussion is; how general are the research results? If there are multipurpose models, and mapping between products and simulators exists, then the results should be applicable to another industrial sector. The ANSI/EIA-632 standard, for example, is general. Examples of sectors with similar needs could be automotive, automation robotics, power plants, heavy machinery, and medical equipment. No study has been made in any sector outside aerospace within this work. It is however reasonable to assume that the results can be generalized to systems realized by hardware and software, but are simulated by software.

7.2 Research results versus the research questions

The result of the prototype configurator as a solution to the industrial problem is covered generally in the results chapter. This section summarizes the response to the explicit sub-questions, outlined in section 2.1.

- To what degree is it possible to modularize the simulation model including parameter sets and other simulation artifacts?

There is no exact limit to how small a module can be, or into how many modules the system can be divided. In practice, each Configuration Item contributes with an overhead in the form of features values, validation, and documentation, so dividing a simulation system too much may become too costly to maintain. In addition, the combinatorial explosion, which is a consequence of many parts, is a disadvantage. The relevant level to divide a simulation model should be into parts that firstly are appropriate to map to the equipments and subsystems of the simulated product. Secondly, it is also relevant to divide a model into an algorithm part and one or several parameter sets. This allows for effective management of common and variant-parts. Mapping onto a product structure reduce process complexity due to, for example, clearer change-flow for the model(s).

- What kinds of variation techniques for simulation model variability, including embedded software, are applicable in the product instantiation process?

Run-time (late) binding with Configuration Parameters provides more flexibility compared to checkout- and compile-time binding. It is clear from the industrial example that as late binding as possible is desirable in most cases. Only specific needs will require other choices, for example confidentiality, which could be a performance or equipment model that is unique to one customer. There are conflicting needs between the simulators that are sold as products and those used in-house. In an in-house development simu-

lator, change-of-configuration must be easier to accomplish. For training simulators (delivered products) there is instead the requirement that only customer-unique features and embedded software must be included, and early binding should then be used.

- How should compatibility constraints for simulation models be specified to be maintainable in a large-scale product line?

The feature modeling technique is suitable for complex products, and specification of constraints should be done by using established modeling methods. The prototype has demonstrated that feature modeling works and is scalable as long as the domain ranges are specified mainly by discrete variables.

- How to specify and build simulation configurator systems based on PDM data where environment models are not part of the product definition?

The simulation and PDM domains are weakly integrated so it may not be suitable to use automated system/tool integration. Manual and semi-automatic mapping of data with the support of XML files for example are appropriate for the near future.

7.3 Contributions

The focus of the research is at the intersection of model based development and product line engineering. The following are the main contributions from this work:

- The main contribution regarding the problem area is the definition of primary and secondary product lines. This description clarifies industrial needs and is a base for further research and development to support Product Line Engineering of 'multi-product lines' and complex industrial products.
- Definition of Modeling Domains is a means of classifying of modeling techniques and analyzing, for example, integration of simulation models/tools. The definition is accepted and used in industry, at Saab Aeronautics, and in the EU research project CRESCENDO. It is used to classify needs, requirements, and means for modeling techniques, integration of simulation tools, and virtual testing.
- Evaluation of the hosted simulation method contributes to both academia and industry. It is used at Saab for development of the 39 Gripen vehicle systems.
- Industrial experience in the aerospace sector from the introduction of the Modelica language, the Simulink tool, and the System Modeling Language SysML contributes to guidance for modeling of complex systems.
- Implementation of a configurator prototype to support customization of simulation models for simulator systems with different implementations, for different purposes and that simulate different product variants. The configurator makes it easy to define a simulation configuration that represents a real product configuration.
- Experience obtained from initial introduction of the configurator support in an existing development environment for aircraft simulators. Integration of the configurator with interfacing systems, including software configuration management and the software build system.
- Evaluation of the configurator prototype based on feedback from users in the department responsible for 39 Gripen simulators, at Saab Aeronautics.

Finally, it was observed that changes in an on-going business should be introduced with caution because the delivery process is vulnerable. This knowledge is of great value and should be a subject for reuse.

7.4 Conclusions

This dissertation covers emerging modeling techniques for large-scale systems modeling and simulation. Experiences have been collected from the introduction of modeling languages to support the development and maintenance of systems and products at Saab Aeronautics. With the more efficient modeling techniques/tools available and increasing computational performance, the number of models tends to increase. Through product line methods, for instance the 'design once approach', models are (re)used for simulations and in products for different purposes. A *multipurpose simulation model* denotes one that is developed and declared for reuse in multiple contexts. Due to the increasing number of models, there is an identified need for configuration support in the set-up of simulations.

A configurator prototype is implemented to investigate how a configurator system can support the design and customization of simulator families. The combination of approaches (product-line / modeling-based) imposes new conditions in the implementation and use of traditional tools and methods, such as change management. Product/sales configurators are available for use by industry, but experience is still limited as regards large-scale multi-product lines. Gaps are identified in the form of a lack of mature toolsets for mixed Product Line / Software Product Line handling. The implemented prototype, based on the Tacton configurator tool, and XML technology for integration to the existing development environment, introduce design automation and customization of simulation. This contributes unique experience and knowledge to the field and points at one possible way forward.

The answer to the main research question is; yes, the principles of product customization are applicable for modular simulation systems in a software-intensive product line context. Existing methods are focused on management for single product lines, and there is a need to continue the development of support for simulator product families and the management of simulator variants and multipurpose models.

7.5 Future work

During demonstration and evaluation of modeling and customization techniques, some areas of further research have been identified.

7.5.1 Enlarged scope of reuse

The focus has been on reuse of simulation models, even if other assets, for instance test cases and validation results are also handled. There is still a potential for a higher degree of reuse of requirements, design descriptions, and other kind of information/documentation associated with the simulation models. To determine the ambition or scope is a balance between benefits and costs of the components used in single or potentially a few products. For assets with a lower degree of commonality, a risk of unnecessary overhead from handling within the more stringent product line procedures

is anticipated. To succeed with a balanced product line from a set of needs, a system-model of the product/simulator architecture is a way forward. It will preferably be based on the SysML modeling technique and inspired by the ANSI/EIA-632 system structure.

7.5.2 Generalization and standardization of methods/tools

Any person who is responsible for methods/tools in an organization would want a standardized, stable, fit-for-purpose tool-chain. With the many new methods / tools available from research and tool vendors, the environment will presumably be "unstable" in the sense that improved methods/tools/versions will change the workplace over time. To gain a more stable development environment, the organization should strive to use established standards.

There is yet no clearly defined standard or established de-facto standard in the field of product line engineering. Leading methods and tools descriptions should be a base for future monitoring of the area. The result from for example the ConIPF project (see section 5.5.4) and its methodology, with the focus on industrial large-scale applications, includes a comprehensive description of tool-independent product line engineering. This methodology would be a good starting point to carry out further work for seeking an appropriate, more standardized, (or de-facto standard) support for simulation products in the future.

7.5.3 Next step at Saab

Saab considers the outcome of the configurator prototype successful and intends to continue the introduction. The results, lessons learned, and parts of the prototype implementation forms a basis for coming steps. For further automation with respect to data and knowledge capture, one suggestion to be tested is the retrieval of existing information. Pugliese, Colombo, and, Spurio [2007] report on a method for retrieval of data from pre-existing components in a repository. In the same manner, information about compatibility constraints in the simulator build-process could be captured through the compilers- outputs (errors/warnings) and be (semi-)automatically fed into the knowledge base.

Automatic generation of Release Notes documents and Configuration Items lists is in operative use. Design of a tool-chain and a workflow including the flow of and decisions on product changes are imminent. The intention is to proceed with the 'design once' approach and a clear responsibility structure within the ‘multi-product organization’ is needed. This includes designated responsibility for simulation models, which are important links between the primary and secondary products. The configurator system highlights those links and provides a new view of the families of models and products.

The use of simulation models and amount of models are expected to increase. Consequently, there is the need for management of models from different points of view. These include model integration and customization, but also the relationship between models and the reality they should represent. With the visual support to manage variability in products and models, engineers gain better insight and control of the simulation configuration, and consequently higher quality of simulation results.

References

Alford, M., White, S., McCay, B., Oliver, D., Tully, C., Holtzman, J… Willey, A. (1992). *Improving the Practice in Computer-Based Systems Engineering*. In Proceedings of the 2nd annual Symposium of the National Council on Systems Engineering, NCOSE, pp. 207–214.

Andersson, H. & Sundkvist, B.G. (2006). *Method and Integrated Tools for Efficient Design of Aircraft Control Systems*. In Proceedings of the 25th International Congress of the Aeronautical Sciences, ICAS. Hamburg, Germany

Andersson, H. (2009). *Aircraft Systems Modeling - Model Based Systems Engineering in Avionics Design and Aircraft Simulation*. Licentiate Thesis. Linköping Studies in Science and Technology. Thesis, 1394: Linköping University Electronic Press

Andersson, H., Weitman, A. & Ölvander, J. (2008). *Simulink as a Core Tool in Development of Next Generation Gripen.* In Proceedings of Nordic Matlab User Conference 2008. Stockholm, Sweden.

ANSI/EIA-632 (1999). *Processes for Engineering a System.* American National Standards Institute, ANSI.

ARINC Specification 610C. (2009). *Guidance for Design of Aircraft Equipment and Software for Use in Training Devices*. Annapolis, MD, USA. Aeronautical Radio, Inc.

Beck, K. & Andres, C. (2005). *Extreme Programming explained: embrace change*. (2. ed.) Boston, MA, USA: Addison-Wesley.

Beuche, D. (2008). *Modeling and building software product lines with pure::variants.* In Proceedings of 12th International Software Product Line Conference, SPLC 2008, art. no. 4626875, pp. 358. doi: 10.1109/SPLC.2008.53

BigLever, (2011). *BigLever Software Gears*. Retr. from: www.biglever.com/solution/product.html

Boehm, B. (2006). *A view of 20th and 21st century software engineering.* In Proceedings of the 28th international conference on Software engineering (ICSE '06). (pp. 12-29). New York, NY, USA: ACM.

Borg, A. (2009). *Processes and Models for Capacity Requirements in Telecommunication Systems*. Linköping Studies in Science and Technology. Dissertations, 1238. Linköping, Sweden: Linköping University Electronic Press.

Bruce, T., Paniscotti, D., Roman, A. & Bhanot, V. (2006). *Using model-driven engineering to complement software product line engineering in developing software defined radio components and applications.* In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA '06). ACM, New York, USA, 846-853. doi: 10.1145/1176617.1176733.

Carloni, L.P., Passerone, R., Pinto, A. & Sangiovanni-Vincentelli, A.L. (2006). Languages and Tools for Hybrid Systems Design. *Foundations and Trends in Electronic Design Automation*. Vol. 1, No. 1-2, pages 1-193, Now Publishers Inc.

Cassandras, C.G. & Lafortune, S. (2008). *Introduction to discrete event systems*. (2. ed.) New York, USA: Springer.

Chang, H.P.K. (2006). *On the Relationship between Feature Models and Ontologies*. Thesis. Waterloo, Ontario, Canada: University of Waterloo

Christen, E. & Bakalar, K. (1999). VHDL-AMS - a hardware description language for analog and mixed-signal applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. 46* (10). Piscataway, NJ, USA. 1263-1272. doi: 10.1109/82.799677

Clements, P. & Northrop, L. (2002). *Software product lines: practices and patterns*. Boston, MA, USA: Addison-Wesley.

CMMI Product Team. (2010). *CMMI for Development*. Version 1.3. (CMU/SEI-2010-TR-033), Software Engineering Institute. Pittsburgh, PA, USA: Carnegie Mellon University. Retrieved from: http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm

CRESCENDO (2010). *Collaborative & Robust Engineering using Simulation Capability Enabling Next Design Optimisation.* European Union Seventh Framework Programme (FP7). Grant agreement n° 234344. Retrieved from: http://www.crescendo-fp7.eu

Crnkovic, I., Asklund, U. & Dahlqvist, A.P. (2003). *Implementing and integrating product data management and software configuration management.* Boston, MA, USA: Artech House.

Dassault Systemes. (2011) *Dymola: Multi-Engineering Modeling and Simulation*. Retrieved from http://www.3ds.com/products/catia/portfolio/dymola

Engel, A., Winokur M., Döhmen, G. & Enzmann, M. (2008). *Assumptions / Promises - Shifting the Paradigm in Systems-Engineering*. In Proceedings of the 18th annual International Symposium of the International Council on Systems Engineering, INCOSE, Utrecht, The Netherlands

European Space Agency. (2003). *Simulation Model Portability Handbook*, EWP-2080, Issue 1, Revision 4: European Space Agency (ESA).

France, R. & Rumpe, B. (2007). *Model-driven development of complex software: A research roadmap*. FoSE 2007: Future of Software Engineering, art. no. 4221611, pp. 37-54. Minneapolis, MN, USA: IEEE Computer Society.

Friedenthal, S., Moore, A.C. & Steiner, R. (2011). *A Practical Guide to SysML: The Systems Modeling Language.* (2nd Edition). San Francisco, CA, USA: Morgan Kaufmann Publications.

Fritzson, P. (2004). *Principles of object-oriented modeling and simulation with Modelica 2.1*. New York, USA: Wiley.

Geiger, R.J. (1982). *Simulator structure*. U.S. Patent No. 4,347,055

GM-VV PDG (2010). *Generic Methodology for Verification and Validation (GM-VV) to Support Acceptance of Models, Simulations, and Data: Reference Manual*. SISO-GUIDE-00X.1-201X-DRAFT-V1.2.3: Simulation Interoperability Standards Organization.

Harrison, N., Gilbert, B., Jeffrey, A., Lauzon, M. & Lestage, R. (2004). *Adaptive and Modular M&S Configuration for Increased Reusability*. Interservice/Industry Training, Simulation and Education Conference. Orlando

Hartmann, H. & Trew, T. (2008). *Using feature diagrams with context variability to model multiple product lines for software supply chains*. In Proceedings of the 12th International Software Product Line Conference, SPLC 2008. 4626836. 12-21. doi: 10.1109/SPLC.2008.15

Herzog, E. & Andersson, H. (2009). *Initial Experience in Contracts Based Systems Engineering*. In Proceedings of the 19th annual international symposium of International Council on Systems Engineering, INCOSE. Singapore

Herzog, E. & Pandikow A. (2005). *SysML: an Assessment*. In Proceedings of the 15th annual International Symposium of the International Council on Systems Engineering, INCOSE

Herzog, E., Andersson, H. & Hallonquist, J. (2010*). Experience from Introducing SysML into a Large Project Organisation.* In Proceedings of the 20th annual international symposium of International Council on Systems Engineering, INCOSE. Chicago, IL, USA

Hopgood, A.A. (2001). *Intelligent systems for engineers and scientists*. (2. ed.) Boca Raton, FL, USA: CRC Press.

Hunt, A. & Thomas, D. (2000). *The pragmatic programmer: from journeyman to master*. Boston, MA., USA: Addison-Wesley.

Hvam, L., Haug, A. & Mortensen, N.H. (2010). *Assessment of Benefits from Product Configuration*. In Proceedings from Workshop on Configuration at the 19th European Conference on Artificial Intelligence - ECAI 2010. Lisbon, Portugal.

Hvam, L., Mortensen, N.H. & Riis, J. (2008). *Product Customization*. Heidelberg/Berlin, Germany, Springer-Verlag.

IBM Rational Rhapsody. (2011). *IBM® Rational® Rhapsody®: Collaborative systems and software design.* Retrieved from: http://www-01.ibm.com/software/awdtools/rhapsody

INCOSE (2010). *Systems Engineering Handbook - A Guide for System Life Cycle Processes and Activities, version 3.2,* International Council on Systems Engineering.

ISO/IEC 15288:2008. (2008). *Systems and software engineering: System life cycle processes*. (2nd edition) Geneva, Switzerland: International Organization for Standardization

ISO 2533:1975. (1975). *Standard Atmosphere.* Geneva, Switzerland: International Organization for Standardization

Jarzabek, S. (2007). *Effective Software Maintenance and Evolution: Reused-based Approach*, New York, USA: Auerbach Publishers Inc.

Johansson, B. (2003). *Model Management for Computational System Design.* Linköping Studies in Science and Technology. Dissertation No. 857. Linköping, Sweden: Linköping University Electronic Press.

Johansson, O., Andersson, H. & Krus, P. (2008). *Conceptual Design Using Generic Object Inheritance*, In Proceedings of the ASME International Design Engineering Technical Conference and Computers and Information in Engineering Conference 2008, IDETC/CIE. Brooklyn, N. Y., USA

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E. & Peterson, A.S. (1990). *Feature-Oriented Domain Analysis (FODA): Feasibility Study*. Software Eng. Inst., Tech. report CMU/SEI-90-TR-21. Pittsburgh, PA, USA: Carnegie Mellon Univ.

Kniberg, H. (2007). *Scrum and xp from the trenches: how we do Scrum,* C4Media Inc.

Krueger, C.W. (2002). Easing the Transition to Software Mass Customization. *Software Product-Family Engineering in: Lecture Notes in Computer Science, Volume 2290/2002*, 178-184. Springer Berlin / Heidelberg. doi: 10.1007/3-540-47833-7_25

Krueger, C.W. (2004). Towards a Taxonomy for Software Product Lines. In van der Linden, F. (Eds.). *Lecture Notes in Computer Science: Software Product-Family Engineering*, 3014. 323-331. Berlin / Heidelberg, Germany: Springer.

Kuhn, R. (2008). *A multilevel approach for aircraft electrical systems design.* Sixth International Modelica Conference, Vol. 1. 95-101. Bielefeld, Germany

Mann, Z.A. (2006). Three public enemies: Cut, copy, and paste. *Computer*, 39 (7). 31-35.

Mar, B. (1992). *Back to basics.* In Proceedings of the 2nd annual international symposium of the National Council on Systems Engineering: NCOSE. pp. 37–43

Matharu, J. (2006). Reusing safety-critical software in aviation, *Electronics systems and software, 4*(1). 32 -35. doi: 10.1049/ess:20060105

Mathworks, Simscape™. (2011). *Simscape – Model and simulate multidomain physical systems.* Retrieved from http://www.mathworks.se/products/simscape

Mathworks, Simulink®. (2011). *Simulink - Simulation and Model-Based Design.* Retrieved from http://www.mathworks.se/products/simulink

Mathworks, Stateflow®. (2011). *Stateflow - Design and simulate state charts.* Retrieved from http://www.mathworks.se/products/stateflow

Mellor, S.J. & Balcer, M.J. (2002). *Executable UML: a foundation for model-driven architecture*. Boston, MA, USA: Addison-Wesley.

MetaCase (2011). *MetaEdit+ Domain-Specific Modeling environment*. Retrieved from http://www.metacase.com/MetaEdit.html

Mittal, S. & Frayman, F. (1989). *Towards a Generic Model of Configuration Tasks*. In 11th International Joint Conference on Artificial Intelligence, Detroit, IL, USA, pp. 1395–1401.Modelica Association. (2011). *Modelica®: Modelica and the Modelica Association.* Retrieved from https://www.modelica.org

Modelisar (2011). *Goals of FMI.* Retrieved from FMI website: http://functional-mockup-interface.org

Moir, I. & Seabridge, A. (2004). *Design and development of aircraft systems: an introduction.* Professional Engineering Publishing, London, UK

MOKA Consortium (2001). *Managing engineering knowledge: MOKA: methodology for knowledge based engineering applications*. London, UK: Professional Engineering Publ.

Muller, G. (2011). *Research in Systems Architecting.* version: 2.0. Retrieved from Gaudí System Architecting homepage: www.gaudisite.nl/ArchitectingResearchMethodPaper.pdf

Munir, Q. & Shahid, M. (2010). *Software Product Line: Survey of Tools*. Student thesis, Linköping, Sweden: Linköping University Electronic Press. uri: urn:nbn:se:liu:diva-57888.

Nagy, I. & Cleophas, L. (2011). *Applying Software Product Line Engineering Techniques to Develop Simulators for Testing and Early System Integration.* Retrieved from iSpring Solutions slide hosting website: http://www.slideboom.com/presentations/436271/Applying-Software-Product-Line-Engineering-Techniques-to-Develop-Simulators-for-Testing-and-Early-System-Integration

NASA, (2008). *Standard for Models and Simulations*, National Aeronautics and Space Administration, NASA-STD-7009, Washington, DC, USA. 20546-0001

Oberkampf, W.L., DeLand, S.M., Rutherford, B.M., Diegert, K.V. & Alvin, K.F. (2002). Error and uncertainty in modeling and simulation. *Reliability Engineering and System Safety*, 75 (3), pp. 333-357.

Object Management Group, SysML™. (2008). OMG Systems Modeling Language (OMG SysML™), Version 1.1, *Object Management Group*, Retrieved from http://www.omg.org/spec/SysML/1.1.

Object Management Group, UML®. (2007). OMG Unified Modeling Language (OMG UML), Infrastructure/Superstructure, V2.1.2 *Object Management Group*, Retrieved from http://www.omg.org/spec/UML/2.1.2

Olhager, J. (2003). Strategic Positioning of the Order Penetration Point. *International Journal of Production Economics. 85* (3). 319-329. doi: 10.1016/S0925-5273(03)00119-1

Oliver, D.W., Kelliher, T.P. & Keegan, J.G. (1997). *Engineering Complex Systems – with models and objects.* McGraw-Hill, New York, USA

Padulo, M. (2009). *Computational Engineering Design under uncertainty – An aircraft conceptual design perspective*. Dissertation. Cranfield University, UK

Park, R.E., et al. (1992). *Software Size Measurement: A Framework for Counting Source Statements*. Software Eng. Inst., Technical Report CMU/SEI-92-TR-20. Pittsburgh, PA, USA. Carnegie Mellon University

Potts, C. (1993). Software Engineering Research Revisited, *IEEE Software*, pp. 19-28.

Praehofer, H. (1996). Object Oriented, Modular Hierarchical Simulation Modeling: Towards Reuse of Simulation Code. *Simulation Modelling Practice and Theory. 4*(4): 5-8. doi: 10.1016/0928-4869(96)83760-8

Pugliese, D., Colombo, G. & Spurio, M.S. (2007). About the integration between KBE and PLM. *Advances in Life Cycle Engineering for Sustainable Manufacturing Businesses*. 131-136. doi: 10.1007/978-1-84628-935-4_23

pure-systems. (2011). *Pure::variants: Variant Management*. Retrieved from http://www.pure-systems.com

Python (2011). *Python Programming Language: Official Website.* Retrieved from http://python.org

Rosenmüller, M. (2011). *Towards Flexible Feature Composition: Static and Dynamic Binding in Software Product Lines*. Dissertation. Magdeburg, Germany. Otto-von-Guericke-Universität

Saab (2011). *Mission Trainer: Operational everyday training for pilots*. Retrieved from: www.saabgroup.com/en/Air/Training_and_Simulation/Training_Media/Mission_Trainer/Features.

Salinesi, C., Mazo, R., Djebbi, O., Diaz, D., Lora-Michiels, A. (2011). *Constraints: The core of product line engineering.* IEEE International Conference on Research Challenges in Information Science (RCIS), Guadeloupe, French West Indies, France.

Salo, O. & Abrahamsson, P. (2007). An iterative improvement process for agile software development. *Software Process Improvement and Practice* 12 (1), pp. 81-100

Schallert, C., Pfeiffer, A. & Bals, J. (2006). *Generator Power Optimisation for a More-Electric Aircraft by Use of a Virtual Iron Bird*. In Proceedings of the 25th International Congress of the Aeronautical Sciences, ICAS. Hamburg, Germany

Schwaber, K. (1995). *SCRUM Development Process*. In Proceedings of OOPSLA'95 Workshop on Business Object Design and Implementation, Austin, TX, USA

Siemens (2011). *Open product lifecycle data sharing using XML: Siemens White Paper*. Siemens PLM Software. Retrieved from: http://www.plm.automation.siemens.com/en_us/products/open/plmxml

Simpson, T.W. (Ed.) (2006). *Product platform and product family design: methods and applications*. (1. ed.) New York, USA: Springer.

Sinnema, M., Krebs, T., Hotz, L., MacGregor, J., Nijhuis, J., Wolter, K. & Deelstra S. (2006). *Configuration in Industrial Product Families: The ConIPF Methodology*. Berlin, Germany: Akademische Verlagsgesellschaft Aka GmbH.

Sivard, G. (2001). *A Generic Information Platform for Product Families*. Dissertation. Stockholm, Sweden: Royal Institute of Technology.

Steinkellner, S. (2011). *Aircraft Vehicle Systems Modeling and Simulation under Uncertainty*. Linköping Studies in Science and Technology. Thesis, 1497: Linköping, Sweden: Linköping University Electronic Press.

Steinkellner, S., Andersson, H., Krus, P. & Lind, I. (2008). *Hosted Simulation for Heterogeneous Aircraft System Development*. In Proceedings of the 26th International Congress of the Aeronautical Sciences, ICAS. Anchorage, AK, USA

Stevens, R. (red.) (1998). *Systems engineering: coping with complexity*. London: Prentice-Hall Europe.

Stone, R.J., Panfilov, P.B. & Shukshunov, V.E. (2011). *Evolution of aerospace simulation: From immersive Virtual Reality to serious games.* In Proceedings of 5th International Conference on Recent Advances in Space Technologies (RAST). 655-662. Istanbul, Turkey: IEEE. doi: 10.1109/RAST.2011.5966921

Tacton Systems AB (2011). *Tacton Configurator*. Retrieved from http://www.tacton.com

Thevenot, H. & Simpson, T. (2006). Commonality indices for product family design: a detailed comparison. *Journal of Engineering Design*, Vol. 17, No. 2, pp. 99–119. doi: 10.1080/09544820500275693

Thunnissen, D. P. (2005). *Propagating and Mitigating Uncertainty in the Design of Complex Multidisciplinary Systems*. Dissertation. Pasadena, CA, USA: California Institute of Technology.

Ulrich, K.T. & Eppinger, S.D. (2008). *Product design and development*. (4. ed.) Boston, MA, USA: McGraw-Hill/Irwin.

van der Linden, F. J., Schmid, K. & Rommes, E. (2007). *Software Product Lines In Action*. Springer-Verlag, Berlin, Germany

van Gurp, J. (2000). *Variability in Software Systems: The Key to Software Reuse*. Licentiate thesis. Karlskrona, Sweden: Blekinge Institute of Technology

Weilkiens, T. (2008). *Systems engineering with SysML/UML: modeling, analysis, design*, Amsterdam, The Netherlands: Morgan Kaufmann OMG Press/Elsevier

Weiss, D. M. & Lai, C. T. R. (1999). *Software Product-Line Engineering: A Family-Based Software Development Process.* Reading, MA, USA: Addison-Wesley

Wickenberg, J., Stamlin, R., Persson, M. & Börjesson, S. (2011). *Challenges for increasing component commonality in platforms.* In Proceedings of the 5th European Conference on Management of Technology, EuroMOT2011, Tampere, Finland, pp. 463-472.

Wikipedia, KBE. *Knowledge-based engineering*. Retrieved from Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Knowledge-based_engineering .

Williamson, K. (2002). *Research methods for students, academics and professionals: information management and systems.* (2. ed.) Wagga Wagga, NSW, Australia: Centre for Information Studies.

World Wide Web Consortium (W3C). (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation. 26 November 2008. Retrieved from: http://www.w3.org/TR/2008/REC-xml-20081126

Wymore, W. (2002). *A System Theoretical Framework for V & V*. In Proceedings of the twelfth annual International Symposium of the International Council on Systems Engineering, INCOSE

Öström, J., Lähteenmäki, J. & Viitanen, T. (2008). F-18 hornet landing simulations using adams and simulink co-simulation. AIAA Modeling and Simulation Technologies Conference and Exhibit, art. no. 2008-6850: AIAA